Markt&Technik

Schneider CPC-Software

Dr. Dobb's Journal
J.E. Hendrix

Small-C Entwicklungssystem

C-Compiler

8080-/Z80-Makro-Assembler · Linker/Loader
Bibliotheksverwalter · Editor/Text-Tools

Für Schneider-Computer
3"-Format

Alle Programme mit Quellcode!

Markt&Technik

Schneider CPC-Software

Dr. Dobb's Journal
J. E. Hendrix

Small-C Entwicklungssystem

C-Compiler

8080-/Z80-Makro-Assembler · Linker/Loader
Bibliotheksverwalter · Editor/Text-Tools

Markt & Technik Verlag Aktiengesellschaft · Hans-Pinsel-Straße 2 · 8013 Haar

Einbandgestaltung: Grafikdesign Heinz Rauner
Druck: Schoder, Gersthofen
Printed in Germany

Inhaltsverzeichnis

1 Das Small-C-Entwicklungssystem

Das Small-C-Entwicklungssystem ist ein komplettes Entwicklungspaket für das Betriebssystem CP/M mit Editor, C-Compiler, Assembler, Linker und vielen weiteren Utilities. Alle Programme sind in Small-C geschrieben und der Quellcode wird mitgeliefert! Dem kundigen Benutzer wird so die Möglichkeit gegeben, sich das Entwicklungssystem nach seinen Wünschen und Erfordernissen zu erweitern und zu modifizieren. Darüber hinaus erhält der Anwender damit eine umfangreiche Sammlung praxisnaher Programme und Tools, die exzellente Beispiele für die effiziente Programmierung in C darstellen. Dadurch ist dieses Produkt eine wertvolle Fundgrube für jeden ernsthaften C-Programmierer.

Das Paket besteht aus drei Teilen, die wiederum aus mehreren Komponenten bestehen:

Small-C-Compiler

Small-C ist ein umfangreicher Subset der Sprache C, dessen Qualität durch diese Programme selbst, die alle in dieser Sprache geschrieben wurden, dokumentiert wird. Der Compiler übersetzt C-Programme in 8080-Assembler; zur Übersetzung des Assembler-Codes kann der mitgelieferte Makro-Assembler oder auch der Microsoft-Assembler M80 verwendet werden.

Small-Mac: 8080-/Z80-Makroassembler und Utilities

Der Makroassembler-Teil besteht aus sechs Programmen. MAC ist der eigentliche Makroassembler. Er arbeitet mit zwei Läufen und erzeugt relokatierbaren 8-Bit-Objektcode im Microsoft-Format. MAC kann mit Hilfe des Programms CMIT an den Befehlssatz der Prozessoren 8080 oder Z80 angepaßt werden. Der Linker LNK verknüpft Objektmodule mit den benötigten Bibliotheksmodulen zu ausführbaren Programmen. Mit dem Loader LGO können Betriebssystemerweiterungen geladen und gestartet werden. Der Bibliotheksmanager LIB verwaltet Bibliotheken mit LNK-kompatiblen Objektmodulen und DREL erlaubt den Dump von LNK- und LIB-Dateien.

Small-Tools: Editor und Text-Tools

Small-Tools ist eine komfortable Sammlung von Text-Tools, die einen weiten Bereich der Textverarbeitung abdecken, von der Eingabe von Programmen und Texten (EDIT) über die Erstellung von Serienbriefen und Formatierung von beliebigen Manuskripten (FORMAT) bis zur Recht-

schreibüberprüfung (englisch) und Sortierung von ASCII-Dateien (SORT/MERGE). Darüber hinaus stehen noch etliche Hilfsprogramme zur Verfügung, die kleinere Aufgaben erledigen, jedoch kombiniert eingesetzt werden können und so zu einem extrem leistungsfähigen Tool werden.

Hardwarevoraussetzungen

Das Small-C-Entwicklungssystem benötigt einen Computer mit dem Betriebssystem CP/M-80, einem Diskettenlaufwerk und mindestens 56 KByte Speicher.

Disketteninhalte

Das Small-C-Entwicklungspaket wird in verschiedenen Diskettenformaten angeboten. Je nach Diskettenkapazität werden eine bis fünf Disketten geliefert. Im folgenden finden Sie eine Übersicht der Inhalte der einzelnen Disketten bei der Auslieferung auf fünf Disketten, bzw. Diskettenseiten. Bei Lieferung auf weniger Disketten wurden die Dateien in geeigneter Weise auf weniger Disketten kopiert. Small-C und Small-Mac werden in lauffähiger Version und auch im Quellcode ausgeliefert, die Small-Tools aus Platzgründen jedoch nur im Quellcode.

Wichtig! Bevor Sie mit dem Small-C-Entwicklungspaket arbeiten, fertigen Sie sich bitte unbedingt erst Kopien aller Disketten an und lagern Sie die Original-Disketten an einem sicheren Ort. Die Erstellung von Kopien ist im Bedienungshandbuch Ihres Computers erläutert. Vergewissern Sie sich vorm Kopieren, daß der Schreibschutz der Original-Disketten aktiviert ist, damit Sie diese nicht versehentlich überschreiben. Arbeiten Sie bitte nur mit den Kopien, damit Sie bei Beschädigungen Ihrer Arbeitsdisketten immer die Möglichkeit haben, auf die Original-Disketten zurückzugreifen.

Jede Datei des Small-C-Pakets wird in den Aufstellungen auf den folgenden Seiten kurz erläutert.

Verzeichnis von Small-C 1 (11 Dateien):

```
AR        C       Programm zum Verwalten von Archivdateien, Quellcode
AR        COM     Programm zum Verwalten von Archivdateien
CC        COM     Small-C-Compiler
CC        SUB     Submit-Datei zum Kompilieren von C-Programmen mit M80 und L80
CCCC      SUB     Submit-Datei zum Kompilieren des Small-C-Compilers mit M80 und L80
CLIB      REL     C-Bibliothek im relokatierbaren Microsoft-Format
HISTORY           Wartungsgeschichte von Small-C
NEWLIB1   SUB     Submit-Datei zum Erstellen einer neuen Bibliothek, Teil 1
NEWLIB2   SUB     Submit-Datei zum Erstellen einer neuen Bibliothek, Teil 2
NEWLIB3   SUB     Submit-Datei zum Erstellen einer neuen Bibliothek, Teil 3
STDIO     H       Standard-Definitionsdatei von Small-C
```

Verzeichnis von Small-C 2 (2 Dateien):

```
CC        ARC    Archivdatei mit dem Quellcode des Small-C-Compilers
CLIB      ARC    Archivdatei mit dem Quellcode der Small-C-Bibliothek
```

Verzeichnis von Small-Mac 1 (10 Dateien):

C	LIB	Small-C-Bibliothek im Small-Mac-Format
C	LST	Small-C-Bibliothek, Modulverzeichnis
C	NDX	Small-C-Bibliothek, Indexdatei
CMIT	COM	Anpassungsutility für Small-Mac
DREL	COM	Dump relokatierbarer Objektdateien
HISTORY		Wartungsgeschichte von Small-Mac
LGO	COM	Lader von Small-Mac
LIB	COM	Bibliotheksverwalter von Small-Mac
LNK	COM	Linker von Small-Mac
MAC	COM	Small-Mac-Makroassembler

Verzeichnis von Small-Mac 2 (34 Dateien):

8080	MIT	Maschinen-Instruktions-Tabelle für 8080
CALL	MAC	Logische und arithmetische Funktionen von Small-C
·CMIT	C	Anpassungsutility, Quellcode
DREL	C	Dump relokatierbarer Objektmodule, Quellcode
END	MAC	Endemodul von Small-C
EXT	H	Include-Datei für Small-Mac
EXTEND	C	Bibliotheksmodul für M.LIB
FILE	C	Bibliotheksmodul für M.LIB
GETREL	C	Bibliotheksmodul für M.LIB
INT	C	Bibliotheksmodul für M.LIB
·LGO	C	Lader, Quellcode
·LIB	C	Bibliotheksverwalter, Quellcode
LINK	MAC	Small-C-Modul zur Verknüpfung mit der Bibliothek
·LNK	C	Linker, Quellcode
M	LIB	Small-Mac-Bibliothek
M	LST	Small-Mac-Bibliothek, Modulverzeichnis
M	NDX	Small-Mac-Bibliothek, Indexdatei
MAC	H	Definitionsdatei für Small-Mac
·MAC	C	Small-Mac-Makroassembler, Quellcode Teil 1
·MAC2	C	Small-Mac-Makroassembler, Quellcode Teil 2
·MAC3	C	Small-Mac-Makroassembler, Quellcode Teil 3
MESS	C	Bibliotheksmodul für M.LIB
MIT	C	Bibliotheksmodul für M.LIB
MIT	H	Include-Datei für Small-Mac
NOTICE	H	Include-Datei für Small-Mac
PUTREL	C	Bibliotheksmodul für M.LIB
REL	C	Bibliotheksmodul für M.LIB
REL	H	Include-Datei für Small-Mac
REQ	C	Bibliotheksmodul für M.LIB
SCAN	C	Bibliotheksmodul für M.LIB
SEEREL	C	Bibliotheksmodul für M.LIB
STDIO	H	Standard-Definitionsdatei von Small-C
WAIT	C	Bibliotheksmodul für M.LIB
Z80	MIT	Maschinen-Instruktions-Tabelle für Z80

Verzeichnis von Small-Tools (44 Dateien):

```
BUF      C     Include-Datei für Small-Tools
CANT     C     Include-Datei für Small-Tools
CATSUB   C     Include-Datei für Small-Tools
CHG      C     Small-Tools-Programm CHANGE
CNT      C     Small-Tools-Programm COUNT
CPT      C     Small-Tools-Programm CRYPT
CPY      C     Small-Tools-Programm COPY
DICT           englisches Wörterverzeichnis für PROOF.SUB
DIGIT    C     Include-Datei für Small-Tools
DTB      C     Small-Tools-Programm DETAB
EDT      C     Small-Tools-EDITor Teil 1
EDT2     C     Small-Tools-EDITor Teil 2
ERROR    C     Include-Datei für Small-Tools
ETB      C     Small-Tools Programm
FMT      C     Small-Tools-TextFORMATierer Teil 1
FMT2     C     Small-Tools-TextFORMATierer Teil 2
FMT3     C     Small-Tools-TextFORMATierer Teil 3
FND      C     Small-Tools-Programm FIND
FNT      C     Small-Tools-Programm FONT
GETWRD   C     Include-Datei für Small-Tools
HISTORY        Wartungsgeschichte von Small-Tools
INDEX    C     Include-Datei für Small-Tools
LST      C     Small-Tools-Programm LIST
MAKSET   C     Include-Datei für Small-Tools
MAKSUB   C     Include-Datei für Small-Tools
MRG      C     Small-Tools-Programm MERGE
OUT      C     Include-Datei für Small-Tools
PAGE     C     Include-Datei für Small-Tools
PAT      C     Include-Datei für Small-Tools
PRINTF   C     Include-Datei für Small-Tools
PROOF    SUB   Submit-Datei für die Rechtschreibprüfung enlischer Texte
PRT      C     Small-Tools-Programm PRINT
SAME     C     Include-Datei für Small-Tools
SCOPY    C     Include-Datei für Small-Tools
SETTAB   C     Include-Datei für Small-Tools
SRT      C     Small-Tools-Programm SORT
STDIO    H     Standard-Definitionsdatei von Small-C
STP      C     Small-Tools-Programm SETUP
STRIP    C     Include-Datei für Small-Tools
TABPOS   C     Include-Datei für Small-Tools
TOOLS    H     Standard-Definitionsdatei von Small-C
TRIM     C     Include-Datei für Small-Tools
TRN      C     Small-Tools-Programm TRANS
XINDEX   C     Include-Datei für Small-Tools
```

2 Der Small-C-Compiler

Der Small-C-Compiler übersetzt eine Untermenge der Sprache C in 8080-Assemblercode, der mit einem Assembler übersetzt werden muß. Er läuft auf Systemen mit 8080, Z80 oder dazu kompatiblen Mikroprozessoren unter dem Betriebssystem CP/M-80 Version 2.2 oder später. Der erzeugte Assemblercode ist kompatibel zum Small-Mac-Makroassembler (der Bestandteil des Small-C-Entwicklungspaketes ist) und zum Microsoft-Assembler M80. Der Small-C-Compiler unterstützt alle Möglichkeiten dieser Assembler und des Small-C-Entwicklungspaketes wie zum Beispiel relokatierbaren Objektcode, das Linken separat kompilierter Module und Verwaltung der relokatierbaren Objektmodule in Bibliotheken.

Der Small-C-Compiler ist selbst in Small-C geschrieben und wird mit Quellcode geliefert. Dadurch kann der Small-C-Compiler modifiziert oder an andere Umgebungen angepaßt werden.

Aufruf des Compilers

Beim Aufruf des Compilers können drei Arten von Parametern angegeben werden, Dateinamen, Umlenkungsanweisungen und Schalter. Das Laufzeitsystem bewerkstelligt die Umlenkung, ohne sie an das Programm weiterzugeben.

Der Small-C-Compiler erhält seine Eingabe standardmäßig von der Standardeingabe *(stdin)*. Die Standardeingabe kann mit der Umlenkungsanweisung < auf eine Datei oder ein Gerät umgelenkt werden. Wenn ein oder mehrere Dateinamen in der Befehlszeile vorhanden sind, nimmt der Small-C-Compiler seine Eingabe nicht von *stdin*, sondern in der angegeben Reihenfolge aus den Dateien. Die Standarderweiterung bei Eingabedateien ist C. Jeder Parameter, der kein Schalter ist (zum Beispiel ein einzelner Bindestrich), wird als Dateiname interpretiert.

Wenn die Eingabe nicht über *stdin* erfolgt, geht die Ausgabe des generierten Assemblercodes in eine Datei mit dem Namen der ersten angegebenen Quelldatei und der Namenserweiterung MAC. Wenn keine Dateinamen angegeben werden, erfolgen Ein- und Ausgabe über die Standardein-/ausgabedateien. Umlenkung (<,>) kann benutzt werden, um die voreingestellten Zuweisungen zu ändern.

Die Schalter bestehen aus einem Bindestrich gefolgt von einem Buchstaben und eventuell weiteren Angaben. Durch einen Bindestrich ohne Buchstaben ("-") oder jeden undefinierten Schalter, bricht der Compiler ab, nachdem der folgende Bedienungshinweis angezeigt worden ist:

```
usage: cc [file]... [-m] [-a] [-p] [-l#] [-o] [-b#]
```

Der Schalter -M (Monitor) bewirkt, daß der Compiler die Kopfzeilen der Funktionen auf dem Bildschirm anzeigt, so daß man jederzeit weiß, wie weit die Kompilierung fortgeschritten ist. Der Schalter ist nützlich, um Fehler innerhalb der Funktionen zu identifizieren.

Der Schalter -A (Alarm) bewirkt, daß bei Fehlern ein Piepser ertönt.

Der Schalter -P (Pause) bewirkt, daß der Compiler nach jedem Fehler anhält. Durch Drücken von RETURN wird die Verarbeitung wieder aufgenommen.

Der Schalter -L# (Listing, # ist ein Dateideskriptor im Bereich 1-9) weist Small-C an, den Quellcode der angegeben Datei zu listen. Wenn als Dateibeschreibung 1 (*stdout*) angegeben ist, wird das Listing mit der normalen Ausgabe gemischt. In diesem Fall geht jeder Quellzeile ein Semikolon voraus. Es gibt kein Listing, wenn der Schalter nicht angegeben ist.

Der Schalter -O (Optimierung) bewirkt, daß der Optimierer des Small-C-Compilers die Programmgröße auf Kosten der Geschwindigkeit reduziert.

Der Ausgabe wird automatisch Anfangs- und Endecode hinzugefügt, damit Programme aus mehreren Dateien getrennt kompiliert und assembliert werden können. Der Schalter -B# existiert nur, wenn der Compiler nicht für die Kombination mit einem Linker konfiguriert ist. In diesem Fall werden Programmteile beim Assemblieren und nicht erst beim Linken kombiniert und es ist notwendig, daß die verwendeten Label nicht doppelt vorkommen. Die Labelnumerierung beginnt mit dem Wert #. Wenn # 0 ist (die Standardeinstellung), wird ein komplettes Programm kompiliert. In diesem Fall wird dem Programm Kopf- und Endecode für die Verbindung mit der Bibliothek bzw. dem Laufzeitsystem hinzugefügt. Ein Wert von 1 bedeutet, daß der erste Teil eines mehrteiligen Programms kompiliert wird; nur der Startcode wird hinzugefügt. Ein Wert zwischen 1 und 9000 definiert einen Zwischenteil; in diesem Fall wird kein Code zur Ausgabe hinzugefügt. Ein Wert von 9000 bedeutet, daß der letzte Teil kompiliert wird; Endecode wird hinzugefügt. Die Werte für # müssen so gewählt werden, daß Konflikte mit Labeln aus anderen Programmteilen vermieden werden.

Beispiele

AUFRUF	KOMMENTAR
`cc`	Kompiliert die Tastatureingaben und gibt den erzeugten Code auf dem Bildschirm aus.
`cc test`	Kompiliert *test.c* in eine Ausgabedatei mit dem Namen *test.mac*.
`cc abc def -o -a`	Kompiliert erst *abc.c*, dann *def.c* in eine Ausgabedatei mit dem Namen *abc.mac*. Es wird auf Codegröße hin optimiert und es ertönt ein Piepser, wenn ein Fehler auftritt.
`cc <prog.c -ll -p`	Kompiliert *prog.c* und gibt den erzeugten Code auf dem Bildschirm aus. Die Quelldatei wird als Kommentar mit in die Ausgabe aufgenommen.
`cc <abc.c >xyz.mac -m`	Kompiliert *abc.c* und erzeugt *xyz.mac*. Die jeweils erste Zeile einer Funktion wird auf dem Bildschirm ausgegeben.

Fehlermeldungen

Der Compiler unterdrückt in einer einfachen Anweisung alle Fehlermeldungen bis auf die erste; in der Praxis funktioniert das gut, da eine Meldung genug ist, um den Fehler in einer Anweisung oder einem Ausdruck zu erkennen.

Wenn der Compiler auf einen Fehler stößt, wird auf dem Bildschirm die fehlerhafte Zeile angezeigt. Ein Pfeil aus den Zeichen /\ wird unter der Zeile ausgegeben und markiert die ungefähre Stelle des Fehlers. Wenn es durch den Schalter -p angefordert wurde, hält der Compiler dann an und wartet auf die Betätigung der Return-Taste.

Einige Programme bewirken, daß eine der internen Tabellen des Compilers überläuft. Wenn dies passiert gibt es zwei Möglichkeiten zu reagieren:

1. Neukompilierung des Compilers, wobei dem zu kleinen Puffer mehr Speicher zugewiesen wird.
2. Änderung des Programms, so daß die Überlaufbedingung beseitigt wird.

Es folgt unten eine alphabetische Liste der Fehlermeldungen mit einer kurzen Erläuterung. Die Erläuterungen beschreiben die Fehlerursachen und Korrekturmöglichkeiten. Gelegentlich kann jedoch eine Fehlermeldung

unter Umständen auftreten, die nicht vom Compiler vorausgesehen werden konnten. Die Fehlermeldungen passen also nicht in allen Fällen exakt zu den wirklichen Fehlern.

MELDUNG ERLÄUTERUNG

`already defined`
: Ein Name wurde auf globaler Ebene oder unter den formalen Argumenten mehr als einmal deklariert.

`bad label`
: Eine *goto*-Anweisung ist mit einem ungültigen Label versehen. Entweder stimmt es nicht mit den C-Namensregel überein oder es fehlt ganz.

`can't subscript`
: Ein Index wird mit etwas verwendet, das weder ein Zeiger noch ein Array ist.

`cannot assign to pointer`
: Eine Initialisierung bestehend aus einer Konstanten oder einem konstanten Ausdruck wird mit einem Zeiger in Verbindung gebracht. Bei Integer-Zeigern (*int*) ist keine Initialisierungsanweisung erlaubt und bei Zeichen-Zeigern (*char*) ist nur eine Ausdrucksliste oder eine Stringkonstante erlaubt.

`global symbol table overflow`
: Die Tabelle für globale Symbole ist übergelaufen. Dies kann behoben werden, indem der Compiler mit höheren Werten für die Symbole NUMGLBS und SYMTBSZ (definiert in CC.DEF) kompiliert wird. NUMGLBS ist die Anzahl der globalen Einträge, die in die Tabelle passen, SYMTBSZ ist die kombinierte Größe (in Bytes) der lokalen und der globalen Tabellen. Ein Kommentar im Quelltext erläutert die Berechnung von SYMTBSZ.

`illegal adress`
: Der Adreß-Operator wurde auf etwas angewendet, was weder eine Variable, ein Zeiger (mit oder ohne Index) noch ein indizierter Arrayname ist.

`illegal argument name`
: Ein Name in der Argumentliste einer Funktiondeklaration entspricht nicht den Regeln zur Bildung eines C-Namens.

`illegal symbol`
: Der Compiler hat ein Symbol gefunden, das nicht den Regeln zur Bildung eines C-Namens entspricht.

`invalid expression`
: Ein Ausdruck besteht aus Teilen, die weder eine Konstante, eine Stringkonstante noch ein gültiger C-Name sind.

`line too long`

Eine Quellzeile ist nach der Bearbeitung durch den Präprozessor länger als LINEMAX (80) Zeichen. Dies kann dadurch behoben werden, daß die Zeile aufgeteilt wird. Es kann jedoch auch der Compiler mit einem größeren Wert für LINEMAX (in CC.DEF) neu kompiliert werden. Es ist zu beachten, daß LINESIZE um 1 größer sein muß als LINEMAX.

`literal queue overflow`

Eine Stringkonstante führt zum Überlauf der Literaltabelle des Compilers. Die Literaltabelle wird dazu verwendet, um Stringkonstanten bis zum Ende einer Funktion zu speichern, erst dann werden sie ausgegeben und wieder gelöscht. Die Literaltabelle kann durch Neukompilierung des Compilers vergrößert werden. Es muß dazu ein größerer Wert für LITABSZ (in CC.DEF) angegeben werden. LITABSZ ist die Größe der Literaltabelle in Bytes. Jede Stringkonstante wird in diesem Puffer mit einem Nullbyte abgeschlossen.

`local symbol table overflow`

Eine lokale Deklaration führt zum Überlauf der Symboltabelle. Die lokale Symboltabelle ist eine Tabelle, in der die Argumente, die einer Funktion übergeben werden, und die lokalen Variablen innerhalb einer Funktion beschrieben werden. Sie wird nach jeder Funktion zur Verwendung durch die nächste gelöscht. Sie muß also groß genug für die Deklarationen einer Funktion sein. Diese Meldung kann beseitigt werden, indem der Compiler mit größeren Werten für NUMLOCS und SYMTBSZ (in CC.DEF) neu kompiliert wird. NUMLOCS ist die Anzahl der Einträge in der Tabelle, SYMTBSZ ist die kombinierte Größe der lokalen und der globalen Tabellen in Bytes. Ein Kommentar im Quelltext erläutert die Berechnung von SYMTBSZ.

`macro name table full`

Eine *#define*-Anweisung führt zum Überlauf der Makronamentabelle. Diese Tabelle kann erweitert werden, indem der Compiler mit größeren Werten für MACNBR und MACNSIZE (definiert in CC.DEF) kompiliert wird. MACNBR ist die Anzahl Namen, die in die Tabelle passen und MACNSIZE ist die Größe der Makronamentabelle in Bytes.

`macro string queue full`

Eine *#define*-Anweisung führt zum Überlauf der Makrostringtabelle. Die Makrostringtabelle ist ein Puffer, in dem die Ersatzstrings von Makronamen gespeichert werden. Die-

ser Fehler kann beseitigt werden, indem der Compiler mit einem größeren Wert für MACQSIZE (in CC.DEF) kompiliert wird. MACQSIZE ist die Größe des Makrostringpuffers in Bytes. Er muß alle Ersatzstrings aufnehmen können, die innerhalb eines Programmes definiert werden. Jeder String wird durch ein Nullbyte abgeschlossen.

`missing token`

Die Syntax erfordert an dieser Stelle eine bestimmte Anweisung, die nicht vorhanden ist.

`multiple defaults`

Eine *switch*-Anweisung enthält mehrere *default*-Label.

`must assign to char pointer or array`

Es wird versucht, etwas anderes mit einer Stringkonstante zu initialisieren als ein Zeichen-Zeiger oder Zeichenarray.

`must be constant expression`

Es wurde dort, wo die Syntax einen konstanten Ausdruck erforderlich macht, etwas anderes gefunden.

`must be lvalue`

Etwas anderes als ein *lvalue* wird im Empfangsfeld eines Ausdrucks verwendet. Ein *lvalue* ist ein Ausdruck (eventuell nur ein Name), der einer Speicherstelle entspricht, die verändert werden kann.

`must declare first in Block`

Eine lokale Deklaration steht in einem Block nach der ersten Anweisung.

`negative size illegal`

Eine Arraydimenionierung ist negativ. Es ist zu beachten, daß auch konstante Ausdrücke als Arraydimensionen verwendet werden können.

`no apostrophe`

Einer Zeichenkonstante fehlt der abschließende Apostroph.

`no closing bracket`

Es trat das Dateiende mitten in einer Funktion auf.

`no comma`

In einer Argument- oder Deklarationsliste fehlt ein Komma.

`no final }`

Es trat das Dateiende mitten in einer zusammengesetzten Anweisung auf.

`no matching #if`

Einem *#else* oder *#endif* ist kein *#ifdef* oder *#ifndef* vorausgegangen.

`no open paren`
: Bei einer Funktion fehlt die linke Klammer, die die Argumentliste einleitet.

`no quote`
: Einer Stringkonstanten fehlt der abschließende Anführungsstrich ("). Stringkonstanten können nicht auf der folgenden Zeile fortgesetzt werden, der abschließende Anführungsstrich muß also auf derselben Zeile stehen, wie der einleitende.

`no semicolon`
: Es fehlt an einer Stelle ein Semikolon, an der von der Syntax eines vorgeschrieben ist.

`not a label`
: Der Name in der *goto*-Anweisung ist zwar definiert, jedoch nicht als Label.

`not allowed with block-locals`
: Eine *goto*-Anweisung tritt in einer Funktion auf, die lokale Deklarationen auf einer niedrigeren Ebene aufweist, als der Funktionskopf. Small-C kann so etwas nicht verarbeiten.

`not allowed with goto`
: Eine lokale Deklaration tritt auf einer niedrigeren Ebene auf, als die Funktion mit einer *goto*-Anweisung. Small-C kann so etwas nicht verarbeiten.

`not allowed in switch`
: Innerhalb einer *switch*-Anweisung tritt eine lokale Deklaration auf. Dies ist bei Small-C nicht erlaubt.

`not an argument`
: Die Elemente einer Argumentliste einer Funktion stimmen nicht mit der entsprechenden Typendeklaration überein.

`not in switch`
: Die reservierten Worte *case* oder *default* stehen außerhalb einer *switch*-Anweisung.

`open error on filename`
: Eine Ein- oder Ausgabedatei kann nicht geöffnet werden.

`open failure on include file`
: Eine Datei, die in einer *#include*-Anweisung angegeben wurde, kann nicht geöffnet werden.

`out of context`
: Eine *break*-Anweisung steht nicht innerhalb einer der Anweisungen *do*, *for*, *while*, *switch* oder ein *continue* steht nicht innerhalb einer der Anweisungen *do*, *for* oder *while*.

`output error`

Während des Schreibens auf Diskette ist ein Fehler aufgetreten. Dies kann an einem Ein-/Ausgabefehler, einer schreibgeschützen Diskette oder ungenügend Diskettenspeicher liegen.

`staging buffer overflow`

Der für einen Ausdruck erzeugte Code übersteigt das Fassungsvermögen des Codepuffers. Der Codepuffer nimmt zeitweise den von einem Ausdruck erzeugten Code auf, so daß er noch im Nachhinein geändert werden kann. Wenn das Ende eines Ausdrucks erreicht ist, wird der Puffer ausgegeben und für den nächsten Ausdruck freigemacht. Dieser Fehler kann behoben werden, indem der Ausdruck in mehrere kleine Ausdrücke aufgeteilt wird oder indem der Compiler mit einem größeren Wert für STAGESIZE (definiert in CC.DEF) neu kompiliert wird. STAGESIZE ist die Größe des Codepuffers in Bytes.

`too many active loops`

Die Verschachtelungstiefe einer Kombination aus den Anweisungen *do*, *for*, *while* und *switch* übersteigt das Fassungsvermögen der While-Tabelle. Diese Meldung stimmt im Falle von *switch* nicht ganz, da es sich dabei nicht um eine Schleifenanweisung handelt. Dieser Fehler kann behoben werden, indem der Wert WQTABSZ (definiert in CC.DEF) vergrößert wird. WQTABSZ ist die Größe der While-Tabelle in Bytes. Sie muß ein Vielfaches von WQSZ sein.

`too many cases`

Die Anzahl der Case-Fälle übersteigt das Fassungsvermögen der Switch-Tabelle. Die Switch-Tabelle kann vergrößert werden, indem der Wert SWTABSZ (definiert in CC.DEF) vergrößert und der Compiler neu kompiliert wird. SWTABSZ ist die Größe der Switch-Tabelle in Bytes. Sie muß ein Vielfaches von SWSIZE sein.

`wrong number of arguments`

Es wurde für ein oder mehrere Argumente einer Funktion bis zum Beginn der eigentlichen Funktion keine Typendeklaration vorgenommen.

Sprachumfang

Hier soll nicht die komplette Syntax von Small-C erläutert werden, es werden im wesentlichen nur die Unterschiede zum vollen C beschrieben. Zum Erlernen von C sollten andere Bücher herangezogen werden, zum Beispiel das Standardwerk über die Programmiersprache C von Kernighan und Ritchie.

Die vom Small-C-Compiler akzeptierte Syntax ist eine Untermenge der Standard-C-Sprache. Innerhalb dieser Untermenge weicht sie nicht von der Standard-Syntax ab, was bedeutet, daß die für Small-C geschriebenen Programme unter Unix kompiliert werden und laufen können. Obgleich die Untermenge begrenzt ist und der Compiler daher nicht jedes schon bestehenden C-Programm akzeptiert, sind die Programme, die man damit schreiben kann, kompatibel zu den vollständigen Compilern.

Grob gesagt sind folgende Eigenschaften des vollen Standard-C nicht verfügbar:

- Fließkomma-Datentypen,
- mehrdimensionale Arrays,
- Strukturen (*struct*) und *unions*,
- Bit-Felder,
- Zeigerarrays,
- *sizeof*,
- und Casts.

Das Ziel war nicht die Unterstützung des vollen C, sondern einer genügend großen Untermenge, die es erlaubt, leistungsfähige C-Programme zu schreiben, die zum Standard-C kompatibel sind.

Small-C versteht folgende Steueranweisungen: *if*, *switch*, *case*, *default*, *goto*, *break*, *continue*, *while*, *for* und *do/while*.

Es werden die folgenden Zuweisungsoperatoren unterstützt: |=, ^=, &=, +=, -=, *=, /=, %=, >>= und <<=.

Small-C kennt die logischen Operatoren || und &&. Die Auswertung erfolgt von links nach rechts und wird beendet, wenn das Ergebnis bekannt ist. Die logischen Operatoren ~ und ! werden unterstützt.

Der Präprozessor des Small-C-Compilers unterstützt die Anweisungen *#include*, *#define*, *#ifdef*, *#ifndef*, *#else*, *#endif*, *#asm* und *#endasm*. Die Anweisungen *#ifdef*, *#ifndef*, *#else* und *#endif* werden auch geschachtelt unterstützt.

Es werden nur die beiden Datentypen Integer (*int*) und Zeichen (*char*) unterstützt. Dies bedeutet, daß der Compiler nur eine Integer-Untermenge der Sprache ist, Fließkommazahlen also nicht verwendet werden können.

Erlaubte Modifizierungen der zwei Grundtypen sind:

1. **name*: erklärt *name* als Zeiger auf ein Element des angegebenen Typs
2. *name [ ]*: syntaktisch identisch zur obigen Zeiger-Erklärung
3. *name [Größe]*: deklariert ein Array mit der angegebenen Größe, wobei jedes Arrayelement vom entsprechenden Typ ist

Innerhalb eines Programms nicht definierte Funktionen werden automatisch als *extern* deklariert.

Die Bibliothek

Small-C unterstützt mit Hilfe seiner Bibliothek und seines Laufzeitsystems (siehe folgendes Kapitel) eine Unix-ähnliche Ein-/Ausgabe-Umleitung und Übergabe der Befehlszeilenparameter. Die Standardausgabe kann an bestehende Dateien angehängt werden (>>). Diskettenverzeichnisse können gelesen werden (<B:). Die Bibliothek von Small-C enthält über 80 Funktionen und ist damit eine fast vollständige Implementation der Standardbibliothek von Unix-Systemen.

Es wird sowohl ASCII- als auch binäre Ein-/Ausgabe unterstützt. Die Funktionen *printf* und *scanf* für die formatierte Ein-/Ausgabe sind Bestandteile der Bibliothek. Direktzugriffsdateien sind auf der CP/M-Satzebene möglich. Programme können für bestimmte Dateien einen beliebig großen Puffer anfordern.

Assemblercode in C-Programmen

Da der Small-C-Compiler Assemblercode erzeugt, können innerhalb von Small-C-Programmen auch Assembleranweisungen verwendet werden. Dies geschieht mittels der Präprozessoranweisungen *#asm* und *#endasm*. Alles was zwischen diesen beiden Anweisungen steht, wird direkt in die erzeugte Assemblerdatei übernommen.

Dateninitialisierung

Man kann globale Variablen, Arrayelemente und Zeiger genau wie im vollen C initialisieren, außer das Symbole nicht zur Initialisierung verwendet werden dürfen. Ohne Initialisierung werden globale Variable standardmäßig auf Null gesetzt.

Es können nur konstante Ausdrücke für die Initialisierung von Variablen und Arrayelementen benutzt werden. Wenn die Größe des Arrays nicht gegeben ist, wird sie durch die Zahl der vorhandenen Initialisierungswerte bestimmt. Zeichenkonstanten mit Backslash-Escapesequenzen sind erlaubt. Wenn mehrfache Initialisierungsparameter vorhanden sind, müssen sie in Klammern eingeschlossen und durch Komma getrennt werden. Wenn zu wenig Initialisierungswerte vorhanden sind, werden die übrigbleibenden Elemente auf Null gesetzt. Bei zu vielen Werten wird eine Fehlermeldung ausgegeben.

Man kann eine Zeichenkette mit Anführungszeichen verwenden, um Zeichenarrays und Zeiger anzulegen. In diesem Fall wird automatisch ein Null-Byte am Ende generiert. Ein Arrayname bezieht sich auf das erste Byte und ein Zeiger enthält die Adresse des ersten Bytes. Wenn keine Arraygröße angegeben ist, wird sie auf die Länge der Zeichenkette plus 1 gesetzt. Wenn die Zeichenkette länger ist als die angenommene Größe, wird die Größe erhöht, damit sie mit der der Zeichenkette übereinstimmen.

Lokale Deklarationen

Der ursprüngliche Compiler akzeptierte lokale Deklarationen überall in einer Funktion und doppelte Deklarationen verursachten Fehler. Diese Version erfordert, daß alle Deklarationen innerhalb eines Blockes zuerst erscheinen. Erlaubt sind mehrfache Deklarationen desselben Symbols. Der lokale Teil der Symboltabelle wird in umgekehrter Reihenfolge durchsucht, um das letzte Vorkommen einer Variablen zuerst zu finden. Beim Verlassen eines Blocks werden die Deklarationen ohne Label innerhalb dieses Blocks aus der Symboltabelle entfernt. Lokale Deklarationen dürfen keine Initialisierungswerte enthalten.

Goto-Anweisung

Es besteht in C-Programmen eigentlich keine zwingende Notwendigkeit für *goto* und man sollte es so weit wie möglich vermeiden, da man dadurch sehr leicht die Programmlogik zerstören kann. Gelegentlich gibt es jedoch Situationen, bei denen man weitschweifigen Code vermeiden kann, ohne die Logik zu verstecken. Es ist ebenfalls nützlich, wenn man existierende Programme nach Small-C konvertieren will. Deshalb gehört *goto* zum Sprachumfang von Small-C.

Es gibt eine Einschränkung bei der Benutzung der *goto*-Anweisung. Da lokale Variable innerhalb eines beliebigen Blocks deklariert werden können, kennt der Compiler die Tiefe des Stapel-Zeigers bei Ziellabeln, die bis jetzt noch nicht definiert wurden, nicht, so daß der Stapel-Zeiger vor der

Verzweigung nicht berichtigt werden kann. Es gibt keine effiziente Methode zur Lösung dieses Dilemma. Daher schließen sich lokale Variable in Blöcken (anders als bei Funktionsbeginn) und *goto*-Anweisungen innerhalb einer Funktion gegenseitig aus.

Die Speicherklasse extern

Die Speicherklasse *extern* darf nur bei globalen Deklarationen angegeben werden. Wenn die LINK-Option wirksam ist, werden solche Objekte zu externen Referenzen für den Assembler erklärt und andere globale Variable werden als Einsprungspunkte definiert. Wenn LINK nicht wirksam ist, werden externe Variable für den Assembler nicht definiert und andere globale Variable werden zwar definiert, aber nicht als Einsprungspunkte. Wenn hinter *extern* nicht *int* oder *char* angegeben wird, wird *int* angenommen.

Übergabe der Argumentzahl

Anders als im Standard-C gibt es bei Small-C für eine Funktions die Möglichkeit festzustellen, wieviele Argumente ihr übergeben wurden. Wenn eine Funktion aufgerufen wird, wird die Zahl der Argumente in den Akkumulator geladen. Die Codesequenz dafür benötigt nur zwei Bytes. Um die Argumentzahl festzustellen, ruft eine Funktion CCARGC auf und weist den zurückgegeben Wert einer Variablen zu. Dies muß als erstes in der Funktion geschehen, da andere Operationen Bibliotheksfunktionen aufrufen können, die den Akkumulator zerstören. Der Compiler erzeugt keinen Code, um die Anzahl der Argument zu laden, wenn CCARGC aufgerufen wird. Da viele Programme die Zahl der Argumente nicht übergeben, übergeht der Compiler dies in Programmen, die die Anweisung *#define NOCCARGC* enthalten. Dies reduziert Programmgröße und Ausführungszeit. Die Funktionen *printf*, *fprintf*, *scanf* und *fscanf* der Small-C-Bibliothek benötigen die Argumentanzahl. NOCCARGC darf also in Programmen, die diese Funktionen aufrufen, nicht verwendet werden.

Symbole und Namen

Symbole (Variablennamen usw.) dürfen von beliebiger Länge sein, es sind jedoch nur die ersten acht Zeichen signifikant; darüber hinausgehende Zeichen sind erlaubt, werden jedoch ignoriert. Die Namen *NameIndex1* und *NameIndex2* werden also vom Compiler beide wie *NameInde* behandelt. Jeder globale Name erzeugt ein Assemblerlabel gleichen Namens. Einige Assembler erlauben nur Label mit einer maximalen Länge von sechs Zeichen und verbieten einige Sonderzeichen und reservierte Symbole. Die Namen der CPU-Register und Assembleranweisungen sind beispielsweise

nicht erlaubt. Es ist also am Besten, Namen so zu wählen, daß Sie nicht mit diesen Bezeichnungen in Konflikt geraten und sie innerhalb der ersten sechs Zeichen eindeutig zu halten. Ebenso sollten Namen vermieden werden, die mit U beginnen, da diese von einigen Systemfunktionen verwendet werden. Die genannten Probleme existieren bei lokalen Variablen nicht, da diese auf dem Stapel gehalten und relativ zum Stapelzeiger adressiert werden. Globale Namen in Kleinbuchstaben werden in Großbuchstaben umgewandelt, bevor sie in die Symboltabelle übernommen werden. Symbole in Klein- oder Großbuchstaben sind daher synonym.

Format der Ausgabedatei

Der Assemblercode, der vom Small-C-Compiler erzeugt wird, wurde so knapp wie möglich gehalten, damit der Compiler auch auf Systemen mit wenig externem Speicherplatz sinnvoll eingesetzt werden kann. Das Assemblerlisting ist deshalb vollkommen unformatiert. Am Anfang einer Zeile stehen keine Leerzeichen, auch nicht wenn der erste Teil ein Befehl ist. Die einzelnen Bestandteile der Assembleranweisungen werden durch ein einziges Leerzeichen getrennt. Es werden in der Ausgabedatei keine Tabs verwendet. Das sieht etwa so aus:

```
CC1:
main::
LXI H,-86
DAD SP
SPHL
LXI H,0
PUSH H
LXI H,4096
PUSH H
CALL auxbuf
POP B
POP B
LXI H,0
DAD SP
PUSH H
LXI H,1
PUSH H
LXI H,8
DAD SP
PUSH H
LXI H,81
PUSH H
LXI H,98
DAD SP
CALL CCGINT##
PUSH H
LXI H,98
DAD SP
CALL CCGINT##
.
.
CC2:DB 117,115,97,103,101,58,32,67,80,84
DB 32,107,101,121,10,0,111,117,116,112
DB 117,116,32,101,114,114,111,114,10,0
.
.
```

Kompatibilität zum Standard-C

Small-C verwendet sowohl Carriage Return als auch Line Feed für das Neue-Zeile-Zeichen (newline). Die meisten vollen C-Implementationen verwenden Line Feed. Ein Neue-Zeile-Zeichen sollte deshalb immer mit der Escapesequenz \n geschrieben werden und nicht als numerische Konstante.

Während der Auswertung eines Ausdrucks nimmt der Small-C-Compiler an, daß jeder nicht deklarierte Name eine Funktion ist. Das Standard-C nimmt dies nur an, wenn der Name als Funktionsaufruf geschrieben ist, das heißt eine Klammer folgt.

Small-C akzeptiert *int arg* um ein formales Argument als Funktionsname zu definieren und *arg(...)*, um die Funktion aufzurufen. Standard-C erfordert *int (*arg)()* bzw. *(*arg)(...)*. Die Standard-C-Syntax sollte verwendet werden, um die Kompatibilität zu wahren.

Jeder Small-C-Ausdruck, der von einer Klammer ("(") gefolgt wird, wird als Funktionsaufruf interpretiert. Wenn diese Möglichkeit verwendet wird, kommt es dadurch zu großen Inkompatibilitäten. Während eine Funktion wie *256()* im Standard-C zurückgewiesen wird, ist sie bei Small-C erlaubt. Es wird daraus der Aufruf einer Routine an der Adresse 256.

Small-C betrachtet Integer-Konstanten immer als dezimale Werte und kennt keine oktalen oder hexadezimalen Konstanten. Führende Nullen in Konstanten sollten in Small-C-Programmen vermieden werden, da sie bei der Portierung eines Programms auf einen C-Compiler mit vollem Sprachumfang Probleme verursachen würden. Diese Zahl würde dann für eine oktale Zahl gehalten.

Small-C erlaubt in oktalen Escapesequenzen (\nnn) nur die Ziffern 0 bis 7, wogegen im vollen C auch die Ziffern 8 und 9 erlaubt sind (Sie werden dort wie die Werte 10 und 11 oktal verarbeitet).

Alle internen arithmetischen Operationen basieren auf 16-Bit-Integern, was bedeutet, daß 8-Bit-*char*-Elemente vor der Verwendung mit Vorzeichen versehen werden. Nicht alle C-Compiler machen dies so, so daß es hier (wie auch zwischen anderen Compilern) zu Inkompatibilitäten kommen kann.

Im Gegensatz zum Standard-C gibt es bei Small-C für eine Funktion die Möglichkeit festzustellen, wieviele Argumente ihr übergeben wurden. Diese Möglichkeit sollte deshalb nur sparsam verwendet werden.

Small-C weist mehrere *case*-Anweisungen mit demselben Wert in einer *switch*-Anweisung nicht zurück, wie das Standard-C macht. Dies würde normalerweise sowieso nur unbeabsichtigt geschehen.

Die *#include*-Anweisung von Small-C erfordert keine spitzen Klammern oder Anführungszeichen für den Dateinamen. Aus Kompatibilitätsgründen sollte jedoch immer *#include <stdio.h>* (für die Standard-Ein-/Ausgabedefinitionen) oder *#include "datei"* (für andere Definitionen) verwendet werden.

Small-C wertet erst die linke Seite einer Zuweisung aus und dann die rechte. Das bedeutet, daß Variablen (wie zum Beispiel Indizes), die verwendet werden, um das Ziel eines zugewiesenen Wertes festzustellen, nicht durch die Auswertung beeinflußt werden. Viele C-Compiler werten erst die rechte Seite aus, so daß damit das Objekt ausgewertet werden kann, dem etwas zugewiesen wird. Es sollten deshalb Zuweisungen vermieden werden, bei denen Variablen auf der linken Seite auch auf der rechten verändert werden.

Small-C wertet Ausdrücke in der Reihenfolge aus, in der sie geschrieben werden. Es ist deshalb bei der Auswertung möglich, das folgende Variablen rechts davon beeinflußt werden. Die Sprachdefinition von C sieht keine bestimmte Auswertungsreihenfolge vor, es ist deshalb am Besten, wenn Ausdrücke vermieden werden, in denen die Reihenfolge der Auswertung die auszuwertenden Werte verändert.

Die Ein-/Ausgabefunktionen von Small-C verwenden zur Identifizierung von Dateien Dateideskriptoren, wogegen die Standard-UNIX-Funktionen Zeiger verwenden. Dieser Unterschied verursacht jedoch solange keine Inkompatibilitäten, wie die Werte, die an die Ein-/Ausgabefunktionen übergeben werden, dieselben sind, die von *fopen* zurückgegeben wurden oder *stdin*, *stdout* oder *stderr* heißen.

Die Formatanweisungen *b* bei den Bibliotheksfunktionen *printf* und *fprintf* und die Formatanweisungen *b* und *u* bei *scanf* und *fscanf* sind nur in Small-C vorhanden. Ihre Verwendung schränkt also die Portabilität ein.

Standard-C kann globale und lokale Variablen initialisieren, Small-C nur globale Variablen, Arrays und Zeiger. Nicht vorbelegte Objekte werden standardmäßig auf binär Null gesetzt.

Anders als im Standard-C kann man nicht mehr als eine Modifizierung pro Deklaration machen, also wird so etwas wie *int(*name)[ ]* nicht akzeptiert. Dies bedeutet keine erhebliche Einschränkung, muß aber erwähnt werden.

3 Die Small-C-Bibliothek

Dieses Kapitel beschreibt die Bibliothek des Small-C-Compilers. Sie wurde unter CP/M 2.2 implementiert und unterstützt den Small-C-Compiler und die von ihm kompilierten Programme. Praktisch sind alle Unix-Funktionen verfügbar, die in einer fremden Umgebung anwendbar sind. Natürlich werden Standarddateien mit Ein-/Ausgabe-Umleitung und Unix-ähnliche Übergabe Befehlszeilenparametern unterstützt. Die Bibliothek wird in zwei Versionen geliefert, eine für den MACRO-80-Assembler von Microsoft (CLIB.REL) und eine für den Small-Mac-Makroassembler (C.LIB), der Bestandteil des Small-C-Entwicklungssystems ist (siehe Kapitel 4).

Bis auf die arithmetische und logische Bibliothek gibt es im Quellcode dieser Bibliothek nur etwa 20 Assemblerzeilen. Daher kann man die Systemfunktionen sehr viel einfacher verstehen und selbst an andere Umgebungen anpassen.

Organisation der Bibliothek

Im allgemeinen wird jede Bibliotheksfunktion getrennt kompiliert und assembliert und wird dann in eine Bibliothek von verschiebbaren Objektmodulen abgelegt, die CLIB.REL (Microsoft-Version) oder C.LIB (Small-Mac-Version) genannt wird. Einige Funktionen, die miteinander in Beziehung stehen, sind in einem einzigen Modul gruppiert. Beispiele sind *printf* und *fprintf* und die Systemfunktionen im Modul CSYSLIB. Beim Linken wird der Linker (L80 oder LNK) angewiesen, die jeweilige Bibliothek (CLIB.REL oder C.LIB) zu durchsuchen, um externe Referenzen aufzulösen. Alles was gebraucht wird, um ein Programm unter CP/M laufen zu lassen wird geladen und mit in die COM-Datei gelinkt. Module, die nicht angesprochen werden, werden auch nicht hinzugeladen. Die Größe der mindestens zu ladenden Funktionen beträgt etwa 5,5 KByte.

Da L80 nicht rückwärts sucht, um ein Modul zu finden, ist die Bibliothek CLIB.REL so organisiert, daß rückwärtige Referenzen nur Module einbeziehen, von denen bekannt ist, daß sie geladen werden müssen. Sonst wird dies alphabetisch organisiert. Der Compiler erzeugt immer eine externe Referenz auf *Ulink*, die nur die Deklaration von *Umain* als extern enthält.

Dies geschieht als erstes in der Bibliothek und erzwingt, daß CSYSLIB geladen wird, was anschließend erfolgt. Das letze Modul in der Bibliothek ist CALL, die arithmetische und logische Bibliothek. Sie wird zuletzt geladen, um den Beginn des freien Speicherplatzes festzustellen.

Namen der Systemfunktionen

Die Namen der Systemfunktionen in der Quelldatei CSYSLIB.C und ihre globalen Variablen begannen früher mit dem Unterstrich um Konflikte mit Benutzer-Funktionen und Variablennamen zu vermeiden. Ältere Versionen von MACRO-80 Versionen akzeptierten diese Namen jedoch nicht als externe Referenzen. Deshalb wurde der Unterstrich durch den Buchstaben U, der "Unterstrich" bedeutet, ersetzt.

Initialisierung und Beendigung des Programms

Der letzte Teil von CALL enthält folgenden Code:

```
Uend: lhld  6           ;bdos Adresse holen
      sphl              ;als Stackbasis benutzen
      lxi   h,Uend      ;Beginn des freien Speicherplatzes holen
      shld  Umemptr##   ;für Speicherzuordnung benutzen
      jmp   Umain##     ;Befehlszeile analysieren, Programm ausführen
      end   Uend
```

Das Label *Uend* bezeichnet das Programmende und den Beginn des freien Speicherplatzes. Durch die letzte Zeile wird angezeigt, daß an dieser Stelle auch die Programmausführung beginnt. Die Linker L80 und LNK erzeugen am Beginn des Benutzerprogramms einen Sprung auf diese Adresse. Diese Logik wird einmal ausgeführt und dann wird dieser Speicherplatz frei für die Benutzung des Programms. Die Routine *Uend* setzt:

1. SP auf die Basis von BDOS, also den CCP überlagernd,
2. setzt *Umemptr* auf den Beginn des freien Speichers und
3. springt nach *Umain*, um die Programmausführung vorzubereiten.

Die Funktion *Uparse()* wird von *Umain()* aufgerufen, um die Analyse und die Ein-/Ausgabe-Umlenkung durchzuführen. Zuerst wird die CP/M-Befehlszeile in einen dynamisch zugeordneten Puffer kopiert und dann mit *Ufield()* untersucht, um Argumente zu isolieren und mit *Uredirect()* um die Zuweisungen von *stdin* und *stdout* zu ändern (Dateierweiterung mit >> wird unterstützt). Wenn eine Umlenkung mißlingt, bricht das Programm nach der Anzeige des Buchstabens "R" für Redirection Error (Umlenkungs-Fehler) ab. Dann werden *argc* und *argv* auf den Stapel gebracht und *main()* wird aufgerufen, um die Programmausführung zu starten. Bei der Rückkehr wird *exit()* mit einem Null-Argument aufgerufen, das den erfolgreichen Abschluß anzeigt. Natürlich kann *exit()* auch vom Programm direkt aufgerufen werden und jeden gewünschten Fehlercode übergeben. Der Fehlercode wird, sofern er ungleich Null ist, als Byte auf dem Bildschirm ausgegeben (7 erzeugt zum Beispiel einen Piepser). Alle offenen Dateien werden geschlossen und ein Warmstart durchgeführt.

Die BDOS-Schnittstelle

Eine BDOS-Schnittstelle ist durch die Funktion *Ubdos()* gegeben. Sie braucht zwei Argumente, als erstes den Funktionscode, der ins C-Register geladen wird und den Wert für das Registerpaar DE. Dann wird die Adresse 5 (CP/M) aufgerufen. Bei Rückkehr enthält HL (das primäre Register für Small-C) den CP/M-Rückkehrcode vom A-Register. Diese einfache Schnittstelle genügt, um die Bibliotheksfunktionen zu realisieren.

Speicherverwaltung

Speicher wird in nicht verketteten, zusammenhängenden Blöcken am Ende des Programms zugeordnet. Jeder Aufruf von *Ualloc()* ordnet einen Block von auf Null gesetztem bzw. nicht initialisiertem Speicher zu, abhänging vom Wert des zweiten Arguments. Die Standardfunktionen *malloc()* und *calloc()* rufen *Ualloc()* auf. Speicher kann *mit free()* oder *cfree()* wieder freigegeben werden. Vorsicht ist geboten, wenn Speicher in umgekehrter Reihenfolge freigegeben wird, wie er zugeordnet worden ist. Bei der Speicherfreigabe wird einfach ein neuer Wert nach *Umemptr* gebracht und alles, was über dieser Adresse liegt, wird als frei angesehen. Eine Funktion mit Namen *avail()* kann aufgerufen werden, um festzustellen, wieviel Platz zwischen *Umemptr* und dem Stapel liegt. Wenn es eine Programm-/Stapelüberschneidung gibt, wird *avail()* entweder, wie angefordert, Null zurückgeben oder das Programm abbrechen, nachdem der Buchstabe "M" für Memory Error (Speicherfehler) angezeigt wurde. Wenn nicht genügend Speicher vorhanden ist, brechen *malloc()* und *alloc()* auf diese Art ab.

Dateiverwaltung

Die Small-C-Bibliotheksfunktionen identifizieren Dateien mittels kleiner Integer-Werte, die Dateideskriptoren genannt werden. Im Gegensatz dazu benutzt die Unix-Standard-Bibliothek einen Zeiger auf eine Dateikontrollstruktur. Unsere Bibliothek benutzt durchweg die Dateideskriptoren, obwohl sie Funktionen der Standard-Ein-/Ausgabe-Bibliothek enthält. Die Auswirkung dieser Unterschiede ist vernachlässigbar, wenn man Dateireferenzen auf die Werte beschränkt, die durch die Funktion *fopen()* zurückgegeben werden und die Symbole *stdin*, *stdout* und *stderr* (definiert in der Datei STDIO.H mit 0, 1 und 2). Das Symbol MAXFILES in der Datei CLIB.DEF entscheidet, wieviele Dateien zur gleiche Zeit offen sein dürfen. Sieben Integer-Arrays werden entsprechend diesem Wert dimensioniert. Im einzelnen sind dies:

`Ustatus[MAXFILES]`

Dies ist ein bitcodiertes Statuswort, das anzeigt, ob eine Datei mit dem entsprechenden Dateideskriptor geöffnet ist; Null bedeutet geschlossen.

Getrennte Bits autorisieren Lesen und Schreiben und es gibt Bits für Dateiende und Fehlerbedingungen.

`Udevice[MAXFILES]`

Enthält einen Wert ungleich Null, der eines der logischen Geräte bei CP/M bedeutet, wenn mit dem entsprechenden Dateideskriptor keine Diskettendatei geöffnet ist.

`Ufcbptr[MAXFILES]`

Beim ersten Öffnen einer Datei mit dem Dateideskriptor wird ein CP/M-Standard-Dateisteuerblock dynamisch zugeordnet und seine Adresse wird hier gespeichert. Wenn die Datei geschlossen wird, bleibt der FCB für späteren Gebrauch erhalten. Es sei daran erinnert, daß das Verfahren für die Speicherzuordnung zu primitiv ist, um wahllos wieder den Speicher freigeben zu können. Ebenso darf das Programm keinen zugeordneten Speicher freigeben, bevor eine Datei geöffnet wird.

`Ubufptr[MAXFILES]`

Wie beim FCB wird auch ein Puffer zugeordnet, wenn eine Datei geöffnet wird; ihre Adresse wird hier abgelegt. Auch die Puffer bleiben erhalten, wenn die Datei geschlossen wird.

`Uchrpos[MAXFILES]`

Dies ist ein Offset zum nächsten Byte, das man aus dem entsprechenden Puffer erhält.

Anmerkung: Der aktuelle Sektor einer Datei wird im Feld für den wahlfreien Zugriff im FCB selbst gespeichert und nur die wahlfreien Lese- und Schreibzugriffe von CP/M werden durchgeführt.

`Udirty[MAXFILES]`

Ein Nullwert zeigt hier an, daß der entsprechende Puffer neue Daten enthält, die noch auf die Diskette geschrieben werden müssen.

`Unextc[MAXFILES]`

Ein Zeichen, das wieder durch *unget()* zur Datei zurück geschrieben worden ist, wird hier gespeichert. Der Wert EOF (definiert in STDIO.H), zeigt ein leeres Feld an, da EOF nicht zurückgeschrieben werden kann.

Bei diesen Arrays wird kein Versuch unternommen, Platz zu sparen, weil Small-C bedeutend mehr Code für Zeichenwerte erzeugt als für Integer. Weil diese Arrays klein sind, würde mehr Code für das Umsetzen dieser Arrays verbraucht werden, als die Zeichenarrays an Platz sparen würden.

Die Funktion *Umode()* wird intensiv benutzt, sowohl für die Überprüfung, ob ein Dateideskriptor gültig ist, als auch, ob die angegebene Datei offen ist. Wenn der Dateideskriptor gültig ist, wird der Status der Datei zurückgegeben, sonst wird Null zurückgegeben.

Die Funktion *Uopen()* wird von *fopen()* und *freopen()* aufgerufen. Es wird versucht, eine Datei mit dem angegebenen Dateideskriptor zu öffnen. Überprüft wird, ob das erste Zeichen im Modus-Argument ein "r" (read, lesen), "w" (write, schreiben) oder "a" (append, erweitern) ist. Wenn der Dateiname CON:, LST:, PUN: oder RDR ist, wird dem Dateideskriptor vor der Rückkehr einfach das entsprechende logische Gerät zugewiesen. Sofern nötig wird sonst der Datei ein FCB und Puffer zugeordnet. Dann wird *Unewfcb()* aufgerufen, um:

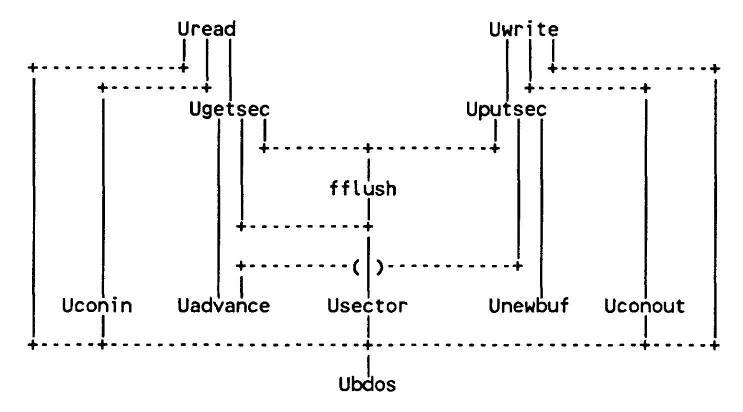
1. die Gültigkeit des Dateinamen zu überprüfen,
2. ihn in Großbuchstaben umzusetzen und
3. den FCB anzulegen.

Schließlich wird *Ubdos()* aufgerufen, um die Datei zu öffnen. Wenn gelesen werden soll, wird der erste Sektor der Datei (128 Bytes, ein CP/M-Satz) automatisch in den Puffer gelesen. Im Falle des Schreibens wird zuerst eine schon bestehende Datei gelöscht und dann eine neue angelegt. Beim Erweitern (Append) wird eine neue Datei angelegt, wenn noch keine existiert. Wenn die Datei schon vorhanden ist, wird sie geöffnet, dann wird die Position des letzten Blocks gesucht und durch wiederholte Aufrufe von *fgetc()* bis EOF (end-of-file) gelesen. Wenn ein Control-Z als Dateiendekennung gefunden wird, wird *Uchrpos* so ausgerichtet, daß ab dieser Position geschrieben werden kann. Obwohl dieses Verfahren vermeidet, die gesamte Datei einzulesen, können bei einer Zeichendatei eingebettete Control-Z-Zeichen übersehen werden. Wenn dem Modus-Zeichen ein "+" folgt (zum Beispiel *r+*), wird ein Aktualisier-Modus (Update) angenommen und *Ustatus* wird auf Lesen und Schreiben gesetzt. Diese neue Merkmal von Unix/C wird von C.D. Perez in "A Guide to the C Library for Unix Users" dokumentiert. Offensichtlich muß man in Unix/C *fseek()* oder *rewind()* aufrufen, wenn man zwischen Lesen und Schreiben wechseln möchte. Die Small-C-Bibliothek erlaubt jedoch ein uneingeschränktes Wechseln zwischen Lesen und Schreiben; jede Operation beginnt mit dem Byte, das dem letzten übertragenen folgt. Da Small-C keine Long-Integer unterstützt, wird auch *fseek()* nicht unterstützt. Stattdessen kann man *cseek()* zur Positionierung auf CP/M-Satzgrenzen benutzen.

Wenn beim Lesen das Dateiende gefunden wird, wird das EOF-Bit in *Ustatus* gesetzt und weiteres Lesen verhindert. Schreiben ist jedoch erlaubt. Das EOF-Bit wird bei erfolgreicher Suche oder durch *rewind* wieder gelöscht. Dateiöffnung im Schreibe- oder Erweiterungsmodus setzt das EOF-Bit automatisch. Dateien kann man entweder durch Öffnen im Erweiterungs-Modus oder durch Öffnen im Modus Lesen/Aktualiseren durch Lesen bis zum Dateiende mit anschließendem Schreiben erweitern.

Unix führt nur binäre Dateiübertragungen durch und das Dateiende wird als Zeiger in der Verzeichnis-Struktur gespeichert. Es ist Aufgabe der Gerätetreiber, zwischen dem Zeichen Neue-Zeile, und der Kombination aus Wagenrücklauf (Carriage Return) und Zeilenvorschub zu unterscheiden. Diesem Verfahren kann jedoch unter CP/M nicht gefolgt werden. Erstens gibt es im CP/M-Dateiverzeichnis keinen Platz für einen Dateiende-Zeiger. Dann muß aus ASCII-Datei-Kompatibilitätsgründen mit anderer CP/M-Software Control-Z als Endekennung verwendet werden und das Zeichen Neue-Zeile muß bei der Ausgabe in eine Kombination aus Carriage Return und Line Feed übersetzt werden (bei der Eingabe umgekehrt). Daher war es erforderlich, ein Unterscheidungsmerkmal zwischen (binären) Byte- und ASCII-Zeichen zu schaffen. Die Absicht der C-Entwickler wäre nicht verletzt worden, wenn man dafür andere, nicht-Unix-kompatible Öffnungs-Modi angegeben hätte. Es wurde jedoch vorgezogen die Standard-Öffnungsprozeduren beizubehalten und stattdessen zwischen den Ein-/Ausgabe-Funktionen zu unterscheiden. In der Small-C-Bibliothek übertragen *read()*, *fread()*, *write()*, und *fwrite()* die Daten binär. Alle anderen Aufrufe (zum Beispiel *getc()*, *fgets()* und so weiter) übertragen ASCII-Zeichen. Dadurch werden Small-C-Programme aufwärtskompatibel zu Unix, ohne die Öffnungsmodi zu ändern.

Binäres Lesen findet das Dateiende nur am Ende des letzten Sektors in der Datei. Das ist notwendigerweise abweichend von Unix, wo das Ende einer Datei auf das Byte genau angegeben werden kann.



In der obigen Abbildung sind die primären Ein-/Ausgabe-Funktionen dargestellt. Linien, die die Funktionsnamen verbinden, kennzeichnen den möglichen Kontrollfluß. Alle Ein-/Ausgabe-Anforderungen gehen entweder durch *Uread()* oder *Uwrite()*. Diese führen nur binäre Datenübertragungen durch, Byte für Byte. Die Logik für zeichenweise Datenübertragung

ist in *fgetc()* und *fputc()*, die wiederum von anderen zeichenorientierten Funktionen aufgerufen werden.

Die Funktionen *Uconin()* und *Uconout()* sind für Übertragungen von und zur Konsole (Bildschirm/Tastatur) zuständig. Für diese Übertragung wird die direkte Konsol-Ein-/Ausgabe von CP/M benutzt. Bei der Ausgabe hat der Terminaltreiber volle Kontrolle über das Programm. Außerdem kann so die Tastatur zuverlässig auf Eingaben des Benutzers abgefragt werden, während noch Daten zum Bildschirm geschrieben werden. Wäre konventionelle Konsol-Ein-/Ausgabe benutzt worden, hätte CP/M auch Kontrollzeichen abgefragt und es wäre reine Glückssache gewesen, wer das Eingabezeichen erhalten hätte. Folge dieser Wahl war, daß die üblichen Tastatureingaberoutinen (*echo*, *rubout* und so weiter) in *Uconin()* und *fgets()* integriert werden mußten. Wenn man sich die Routinen ansieht, hielten sich die Kosten in Grenzen.

Wie ihre Namen schon vermuten lassen übertragen *Ugetsec()* und *Uputsec()* einen Sektor zwischen Puffer und Diskette. Sie werden aufgerufen, wenn die Puffer voll werden oder leer sind. Einen Sektor zu holen setzt voraus, daß der Puffer zuerst auf die Diskette geschrieben wird, wenn er neue Daten enthält. Als nächstes wird die wahlfreie Satzzahl des FCB (RRN) durch Aufruf von *Uadvance()* weitergestellt. Schließlich werden die Daten durch Aufruf von *Usector()* (wobei *Ubdos* aufgerufen wird) übertragen. Der Dateiendestatus wird gesetzt, wenn der Versuch mißlingt. Einen Sektor zu schreiben unterscheidet sich vom Lesen um mehr als nur die Umkehrung des Datenflusses. Die Reihenfolge der Aufrufe von *Usector()* und *Uadvance()* wird umgekehrt, da der RRN jetzt die Position in der Datei beschreibt, wohin der neu gefüllte Puffer geschrieben werden sollte. Nach der Datenübertragung wird *Unewbuf()* aufgerufen um den Puffer in Erwartung weiterer *Uwrite()*-Aufrufe mit Control-Z-Zeichen aufzufüllen.

Die *Ugetsec()*-Funktion erlaubt das Lesen von Diskettenverzeichnissen. Ein Verzeichnis wird als ASCII-Datei mit Dateinamen angesehen, einer pro Zeile. Ein Verzeichnis wird durch eine Laufwerksangabe ohne Dateiname angezeigt. Durch die Bezeichnung B: erhält man zum Beispiel das Verzeichnis des Laufwerks B. X: bedeutet das aktuelle Verzeichnis. *Fopen()* und *freopen()* akzeptieren diese Verzeichnisnamen genau wie jeden anderen Namen. Verzeichnisnamen können auch benutzt werden, um die Standardeingabe umzulenken. Verzeichnisdateien können nur gelesen werden; beim Schreiben erhält man einen Fehler. *Isatty()* antwortet mit *YES* bei Verzeichnisdateien, *cseek()* gibt *EOF* zurück, *fflush()* macht nichts und *ungetc()* arbeitet wie üblich. Diese Merkmale benötigen 0.3KB, die durch Löschen von *#define DIR* in CSYSLIB.C vor der Kompilierung der Small-C-Bibliothek entfernt werden können.

Die Funktionen

Die meisten Funktionen auf Benutzerebene sind nach ihren Unix-Gegenstücken angelegt worden. Einige sind jedoch extra für diese Bibliothek geschaffen und als Small-C-Funktionen gekennzeichnet worden. Hier einige wichtige Symbole, die in STDIO.H definiert werden:

```
#define stdin  0  /* Dateideskriptor für Standardeingabe */
#define stdout 1  /* Dateideskriptor für Standardausgabe */
#define stderr 2  /* Dateideskriptor für Standardfehlerausgabe */
#define ERR   -2  /* Rückgabewert bei Fehlerbedingung */
#define EOF   -1  /* Rückgabewert bei Dateiende */
#define NULL   0  /* Wert eines Null-Zeichens */
```

Ein-/Ausgabe-Funktionen

`fopen(name, mode) char *name, *mode;`

Diese Funktion versucht die Datei zu öffnen, die durch die mit Null abgeschlossene Zeichenkette *name* angegeben wird. *Mode* zeigt auf eine Zeichenkette, die anzeigt, wie die zu öffende Datei benutzt werden soll. Die Werte für mode sind "*r*" (lesen, read), "*w*" (schreiben, write) und "*a*" (erweitern, append).

Bei "*r*" wird eine schon existierende Datei für die Eingabe geöffnet, "*w*" legt entweder eine neue Datei an oder löscht eine schon bestehende, so daß vom Anfang der Datei geschrieben wird. Bei "*a*" kann man am Ende einer Datei noch weiter schreiben (oder am Anfang einer neuen). Zusätzlich kann man eine Datei auch noch aktualisieren (sowohl schreiben als auch lesen). Die Parameter dafür heißen "*r+*" (aktualisieren lesen), "*w+*" (aktualisieren schreiben) und "*a+*" (aktualisieren erweitern).

Diese Parameter verhalten sich bei einer Dateieröffnung genauso, wie ihre Gegenstücke ohne Aktualisierung, sie erlauben es jedoch, daß zwischen *read()* und *write()* gewechselt werden kann, indem man abwechselnd Ein- und Ausgabe-Funktionen aufruft. Wenn das Programm kein *seek* oder *rewind* durchführt, wird der nächste Schreib- oder Lesevorgang an der Stelle beginnen, an der der vorige geendet hatte.

Wenn die Datei erfolgreich eröffnet wurde, gibt *fopen()* einen Dateideskriptor für die offene Datei zurück, sonst NULL. Dieser Dateideskriptor wird dann für alle folgenden Ein-/Ausgabefunktionen dieser Datei verwendet. Nur die Standard-Dateien können benutzt werden, ohne vorher *fopen()* aufzurufen.

`freopen(name, mode, fd) char *name, *mode; int fd;`

Diese Funktion schließt eine zuvor geöffnete Datei, die durch den Dateideskriptor *fd* angegeben wird, und öffnet eine neue, deren Name in der

mit Null beendeten Zeichenkette *name* steht. *Mode* zeigt auf die Zeichenkette, die den Öffnungsmodus angibt (die den Modi von *fopen()* entsprechen). Zurückgegeben wird bei Erfolg der ursprüngliche Dateideskriptor *fd* oder NULL, wenn die alte Datei nicht geschlossen oder wenn die neue nicht geöffnet werden konnte. Man muß jedoch darauf achten, daß man in diesem Fall nicht zwischen Erfolg und Mißerfolg unterscheiden kann, da der Dateideskriptor *fd* für die Standardeingabe Null ist.

`fclose(fd) int fd;`

Diese Funktion schließt die angegebene Datei. Wenn sich noch neue Daten im Puffer der Datei befinden, werden sie zuerst in die Datei geschrieben. Bei Erfolg wird NULL, bei Fehler ein Wert ungleich Null zurückgegeben.

`fgetc(fd) int fd;` (alias getc)

Diese Funktion gibt das nächste Zeichen aus der Datei zurück, die durch *fd* angegegben wird. Wenn keine weiteren Zeichen mehr in der Datei vorhanden sind oder wenn ein Fehler auftritt, wird EOF zurückgegeben. Das Dateiende wird entweder durch das implementationsabhängige Dateiende-Zeichen gefunden oder durch das physikalische Ende der Datei.

`ungetc(c, fd) char c; int fd;`

Diese Funktion stellt logisch (nicht physikalisch) das Zeichen *c* in die Datei zurück, die durch *fd* angegegben wird. Beim nächsten Lesen aus dieser Datei wird dieses Zeichen zuerst geholt. Nur ein Zeichen auf einmal kann so in Wartestellung gehalten werden. Bei Erfolg wird das Zeichen selbst zurückgegeben oder EOF, wenn ein vorher zurückgestelltes Zeichen wartet oder wenn *c* den Wert EOF hat. EOF kann nicht in eine Datei zurückgestellt werden. Bei einem *seek-* oder *rewind*-Vorgang wird das zurückgestellte Zeichen vergessen.

`getchar()`

Diese Funktion entspricht *fgetc(stdin)*.

`fgets(str, sz, fd) char *str; int sz, fd;`

Diese Funktion liest bis zu *sz-1* Zeichen aus einer Datei, die durch *fd* gekennzeichnet wird, ab Adresse *str*. Durch Übertragung des Zeichens Neue-Zeile wird die Eingabe beendet. Ein Null-Zeichen wird nach dem ersten Neue-Zeile-Zeichen oder an die letzte Stelle angehängt, wenn Neue-Zeile nicht gefunden wird. *Fgets()* gibt bei Erfolg *str* zurück, bei Dateiende oder Fehler NULL.

```
fread(ptr,sz,cnt,fd) char *ptr; int sz, cnt, fd;
```

Diese Funktion liest aus der durch *fd* angegebenen Datei *cnt* Datensätze in einer Länge von *sz* Bytes ab Adresse *ptr*. Die Anzahl der tatsächlich gelesenen Sätze wird an der Aufrufer zurückgegeben. Dies kann weniger als *cnt* sein, wenn vorher das Dateiende gefunden wurde. Diese Funktion führt eine binäre Übertragung durch, Sequenzen aus Carriage-Return und Linefeed werden nicht in Neue-Zeile-Zeichen umgewandelt und auf ein Dateiende-Byte wird nicht geachtet. Erkannt wird nur das physikalische Ende einer Datei. Man sollte *feof()* aufrufen, um sicherzugehen, daß keine Daten mehr da sind und *ferror()*, um Fehlerbedingungen zu entdecken.

```
read(fd, ptr, cnt) int fd, cnt; char *ptr;
```

Diese Funktion liest aus einer durch *fd* angegebenen Datei *cnt* Bytes ab Adresse *ptr*. Die Anzahl der tatsächlich gelesenen Bytes wird an der Aufrufer zurückgegeben. Dies kann weniger als *cnt* sein, wenn vorher das Dateiende gefunden wurde. Diese Funktion führt eine binäre Übertragung durch, Sequenzen aus Carriage-Return und Linefeed werden nicht in Neue-Zeile-Zeichen umgewandelt und auf ein Dateiende-Byte wird nicht geachtet. Erkannt wird nur das physikalische Ende einer Datei. Man sollte *feof()* aufrufen, um sicherzugehen, daß keine Daten mehr da sind und *ferror()*, um Fehlerbedingungen zu entdecken.

```
gets(str) char *str;
```

Diese Funktion liest Zeichen von *stdin* an die Adresse *str*. Die Eingabe wird beendet, wenn ein Neue-Zeile-Zeichen gefunden wird, das Zeichen selbst wird jedoch nicht übertragen. Ein Null-Zeichen beendet die Eingabezeichenkette. *Gets()* gibt bei Erfolg *str* zurück, bei Dateiende oder Fehler NULL. Da diese Funktion eine beliebige Datenmenge übertragen kann, muß man die Größe der Eingabezeichenkette prüfen, um sicherzustellen, daß der zugewiesene Platz nicht überschritten wurde.

```
feof(fd) int fd;
```

Diese Funktion gibt einen Wert ungleich Null zurück, wenn die durch *fd* angegebene Datei ihr Ende erreicht hat. Sonst wird NULL zurückgegeben.

```
ferror(fd) int fd;
```

Diese Funktion gibt einen Wert ungleich Null zurück, wenn bei der durch *fd* angegebenen Datei seit ihrer Öffnung eine Fehlerbedingung aufgetreten ist. Sonst wird NULL zurückgegeben.

`clearerr(fd) int fd;`

Diese Funktion löscht den Fehlerstatus der Datei, die durch *fd* gekennzeichnet wird.

`fputc(c, fd) char c; int fd;` (alias putc)

Diese Funktion schreibt das Zeichen *c* auf die durch *fd* gekennzeichnete Datei. Bei Erfolg wird das Zeichen selbst zurückgegeben, sonst EOF. Wenn *c* ein Neue-Zeile-Zeichen ist, wird ein Carriage-Return/Linefeed-Paar geschrieben.

`putchar(c) char c;`

Diese Funktion entspricht *fputc(c,stdout)*.

`fputs(str, fd) char *str; int fd;`

Diese Funktion schreibt Zeichen ab Adresse *str* zur Datei *fd*. Es werden solange aufeinanderfolgende Zeichen geschrieben, bis ein Null-Byte gefunden wird. Das Null-Byte wird nicht geschrieben und ein Neue-Zeile-Zeichen wird nicht hinzugefügt.

`puts(str) char *str;`

Diese Funktion arbeitet wie *fputs(str, stdout)*, nur daß noch ein Neue-Zeile-Zeichen angefügt wird.

`fwrite(ptr,sz,cnt,fd) char *ptr; int sz, cnt, fd;`

Diese Funktion schreibt auf die durch *fd* gekennzeichnete Datei *cnt* Sätze in der Länge von *sz* Bytes ab Speicheradresse *ptr*. Die Anzahl der geschriebenen Sätze wird zurückgegeben. Durch eine Fehlerbedingung kann die Zahl der geschrieben Sätze kleiner als *cnt* sein. Man sollte *ferror()* aufrufen, um Fehlerbedingungen festzustellen. Diese Funktion führt eine binäre Übertragung durch; Neue-Zeile-Zeichen werden nicht in Carriage-Return/Linefeed-Sequenzen umgewandelt.

`write(fd, ptr, cnt) int fd, cnt; char *ptr;`

Diese Funktion schreibt auf die durch *fd* gekennzeichnete Datei *cnt* Bytes ab Speicheradresse *ptr*. Die Anzahl der geschriebenen Bytes wird zurückgegeben. Durch eine Fehlerbedingung kann die Zahl der geschrieben Bytes kleiner als *cnt* sein. Man sollte *ferror()* aufrufen, um Fehlerbedingungen zu finden. Diese Funktion führt eine binäre Übertragung durch; Neue-Zeile-Zeichen werden nicht in Carriage-Return/Linefeed-Sequenzen umgewandelt.

`fflush(fd) int fd;`

Diese Funktion erzwingt, daß alle systemgepufferten Änderungen in die Datei geschrieben werden. Gewöhnlich werden die in eine Datei zu schreibenden Daten in einem Puffer gehalten, bis 1.) der Puffer voll wird, 2.) der Puffer für einen anderen Datensektor gebraucht wird oder 3.) die Datei geschlossen wird. *Fclose()* ruft diese Funktion auf. Bei Erfolg gibt *fflush()* NULL zurück, bei Fehler EOF.

`cseek(fd, offset, from) int fd, offset, from;`

Diese Small-C-Funktion stellt die durch *fd* gekennzeichnete Datei auf den Anfang des 128 Byte-Satzes, der *offset* Sätze vom ersten Satz, aktuellen Satz oder Dateiende entfernt ist, je nachdem ob *from* 0, 1 oder 2 ist. Nachfolgendes Lesen oder Schreiben beginnt an diesem Punkt. Bei Erfolg wird NULL zurückgegeben, sonst EOF.

`rewind(fd) int fd;`

Diese Funktion stellt die durch *fd* gekennzeichnete Datei auf den Anfang. Das entspricht einem *cseek* nach dem ersten Satz der Datei. Bei Erfolg wird NULL, sonst EOF zurückgegeben.

`ctell(fd) int fd;`

Diese Small-C-Funktion gibt die Nummer des aktuelle Satzes einer Datei zurück, die durch *fd* angezeigt wird. Der zurückgegebene Wert ist der Abstand des aktuellen 128-Byte-Satzes vom ersten Satzes in der Datei. Wenn *fd* keiner Diskettendatei zugewiesen war, wird -1 zurückgegeben.

`unlink(name) char *name;` (alias delete)

Diese Funktion löscht die Datei, die durch die mit Null beendete Zeichenkette in *name* angegeben ist. Bei Erfolg wird NULL, sonst ERR zurückgegeben.

`rename(old, new) char *old, *new;`

Diese Small-C-Funktion ändert den mit *old* angegeben Dateinamen in den mit *new* angegebenen. Bei Erfolg wird NULL sonst ERR zurückgegeben.

`auxbuf(fd, size) int fd, size;`

Diese Funktion ordnet *fd* einen Hilfspuffer der Größe *size* Bytes zu. Bei Erfolg wird NULL, bei Fehler ERR zurückgegebenen. *fd* muß geöffnet sein. *Size* muß größer als Null und kleiner als der verfügbare Speicherplatz sein. Wenn *fd* ein Gerät ist, wird zwar der Puffer zugeordnet, aber igno-

riert. Zusätzlicher Puffer ist nützlich, wenn die Bewegung des Diskettenkopfes oder während sequentieller Operationen der häufige Wechsel zwischen Laufwerken verringert werden soll. Wenn ein Hilfspuffer einmal zugeordnet ist, bleibt er während der Programmausführung zugeordnet, auch wenn *fd* geschlossen wird. Beim zweiten Aufruf dieser Funktion mit dem gleichen *fd* wird ERR zurückgegeben, was jedoch keine Auswirkungen hat. Abwechselnde Schreib-/Leseoperationen oder *seeks* führen zu nicht vorhersehbaren Ergebnissen. *Ungetc( )* wird jedoch normal arbeiten.

```
iscons(fd) int fd;
```

Diese Small-C-Funktion gibt einen Wert ungleich Null zurück, wenn *fd* der Konsole zugeordnet ist, sonst NULL.

```
isatty(fd) int fd;
```

Diese Funktion gibt einen Wert ungleich Null zurück, wenn *fd* einem Gerät statt einer Datei zugewiesen ist, sonst NULL.

Funktionen für formatierte Ein-/Ausgabe

```
printf(str, arg1, arg2, ...) char *str;
```

Diese Funktion schreibt auf die Standardausgabe eine formatierte Zeichenkette, die aus einer mit Null abgeschlossenen Zeichenkette *str* besteht, die an spezifizierten Punkten mit den Zeichenkettenäquivalenten der Argumente arg1, arg2... verknüpft ist. Zurückgegeben wird die Gesamtanzahl der geschriebenen Zeichen. Die Zeichenkette *str* wird auch als Kontrollzeichenkette bezeichnet. Sie muß angegeben werden, die anderen Parameter sind optional. Die Kontrollzeichenkette enthält gewöhnliche Zeichen und Zeichengruppen, die auch als Umwandlungsanweisung bezeichnet werden. Jede dieser Anweisungen sagt *printf( )*, wie das entsprechende Argument in eine Zeichenkette für die Ausgabe umgewandelt wird. Bei der Ausgabe ersetzt das umgewandelte Argument die Umwandlungsanweisung. Das Zeichen % signalisiert den Beginn einer Umwandlungsanweisung und einer der Buchstaben *b*, *c*, *d*, *o*, *s*, *u* oder *x* beendet sie.

In der angegebenen Reihenfolge und ohne Leerstellen dazwischen darf ein Minuszeichen (-) auftreten, eine dezimale Integer-Konstante (nnn) und/oder ein Dezimalbruch (.mmm). Diese Unterfelder sind alle optional. Tatsächlich kann man häufig Umwandlungsanweisungen ohne sie antreffen. Das Minuszeichen bedeutet, daß die aus dem Argument durch Anwendung einer bestimmten Umwandlungsanweisung erzeugte Zeichenkette im Ausgabefeld linksbündig erscheinen muß.

Die dezimale Integer gibt die Mindestbreite dieses Feldes (in Zeichen) an. Wenn mehr Platz gebraucht wird, wird er benutzt, mindesten wird jedoch die angegebene Zahl der Positionen erzeugt. Der Dezimalbruch wird benutzt, wenn das umzuwandelnde Argument selbst eine Zeichenkette ist (genauer, die Adresse einer Zeichenkette). In diesem Fall gibt der Dezimalbruch an, wieviele Zeichen maximal aus der Zeichenkette genommen werden können. Wenn in dem Ausdruck kein Dezimalbruch vorhanden ist, wird die gesamte Zeichenkette benutzt.

Der Buchstabe am Ende gibt die Art der anzuwendenden Umwandlung auf das Argument an. Er kann einer der folgenden sein:

b Das Argument wird als Integer ohne Vorzeichen betrachtet und für die Ausgabe ins binäre Format umgewandelt. Führende Nullen werden nicht erzeugt. Diese Spezifikation gibt es nur in Small-C und bei der Anwendung sollte man das berücksichtigen.

c Dieses Argument wird als Zeichen ohne Umwandlung ausgegeben. In diesem Fall wird das höherwertige Byte ignoriert.

d Das Argument wird als Integer mit Vorzeichen angesehen und und für die Ausgabe in eine dezimale Zeichenkette (möglicherweise mit Vorzeichen) umgewandelt. Führende Nullen werden nicht erzeugt. Das Vorzeichen steht am weitesten links; es ist leer für positive und "-" für negative Zahlen.

o Das Argument wird als Integer ohne Vorzeichen gesehen und für die Ausgabe ins oktale Format umgewandelt. Führende Nullen werden nicht erzeugt.

s Das Argument ist die Adresse einer mit Null abgeschlossenenen Zeichenkette, die ausgegeben wird wie sie ist. Die angegebene Ausrichtung, Mindestbreite und maximale Größe werden jedoch berücksichtigt.

u Das Argument wird als Integer ohne Vorzeichen gesehen und für die Ausgabe in eine dezimale Zeichenkette ohne Vorzeichen umgewandelt. Führende Nullen werden nicht erzeugt.

x Das Argument wird als Integer ohne Vorzeichen gesehen und für die Ausgabe ins hexadezimale Format umgewandelt. Führende Nullen werden nicht erzeugt.

Wenn dem % irgendetwas anderes als eine gültige Angabe folgt, wird das ignoriert und das nächste Zeichen wird ohne Änderung ausgegebenen. Durch %% erhält man %.

Printf() durchsucht die Kontrollzeichenkette von links nach rechts und gibt alles auf *stdout* aus, bis ein % gefunden wird. Dann wird die folgende Umwandlungsanweisung ausgewertet und auf das erste Argument angewendet,

das der Kontrollzeichenkette folgt. Die sich ergebende Zeichenkette wird nach *stdout* geschrieben. Anschließend werden wieder Daten aus der Kontrollzeichenkette geschrieben bis eine andere Umwandlungsanweisung gefunden wird, die dann auf das zweite Argument angewendet wird. Diese Prozedur geht weiter, bis die Kontrollzeichenkette erschöpft ist. Als Ergebnis erhält man eine formatierte Ausgabe, die aus Literalen und variablen Daten besteht.

`fprintf(fd,str,arg1,arg2, ...) int fd; char *str;`

Diese Funktion arbeitet wie *printf()*, nur daß die Ausgabe zur durch *fd* angegebenen Datei geht.

`scanf(str, arg1, arg2, ...) char *str;`

Diese Funktion liest eine Folge von Feldern von der Standardeingabe, wandelt sie in internes Format entsprechend der in der Kontrollzeichenkette *str* enthaltenen Umwandlungsanweisung um und speichert sie an den durch die Argumente *arg1, arg2...* angegebenen Plätzen. Zurückgegeben wird die Zahl der gelesenen Felder. Ein Feld bei der Eingabe ist eine zusammenhängende Kette von Grafikzeichen. Beendet wird es durch das nächste Leerzeichen (Leerstelle, Tab oder Neue-Zeile) oder, wenn die Umwandlungsanweisung eine maximale Feldlänge angibt, wenn diese Feldlänge erschöpft ist. Normalerweise beginnt ein Feld mit dem ersten Grafikzeichen nach dem vorigen Feld; demnach werden Leerzeichen übergangen. Da das Neue-Zeile-Zeichen übersprungen wird, wenn das nächste Feld gesucht wird, liest *scanf()* soviele Eingabezeilen wie erforderlich sind, um die Zahl der Umwandlungsanweisungen in der Kontrollzeichenkette abarbeiten zu können. Jedes der Argumente, das der Kontrollzeichenkette folgt, muß eine Adresse ergeben.

Die Kontrollzeichenkette enthält Anweisungen für die Umwandlung und Leerzeichen (die ignoriert werden). Jede Umwandlungsanweisung sagt *scanf()*, wie das entsprechende Feld in internes Format umgewandelt werden soll und jedes *str* folgende Argument gibt die Adresse an, wo der Wert gespeichert werden soll.

Das Zeichen % signalisiert den Beginn einer Umwandlung und einer der Buchstaben *b*, *c*, *d*, *o*, *s*, *u* oder *x* beendet sie.

In der angegebenen Reihenfolge und ohne Leerstellen dazwischen kann ein Stern (*) auftreten und/oder eine dezimale Integer-Konstante. Diese Unterfelder sind beide optional. Tatsächlich kann man häufig Umwandlungsanweisungen antreffen, die keines von beiden benutzen. Der Stern bedeutet, daß das entsprechende Feld bei der Eingabe übersprungen wird. Aus-

lassungsanweisungen haben keine Argumente. Das numerische Feld gibt die maximale Feldbreite in Zeichen an. Wenn vorhanden, wird das Feld bei der angegebenen Zahl der zu durchsuchenden Zeichen beendet, sogar wenn kein Leerzeichen gefunden wurde. Wenn jedoch ein Leerzeichen gefunden wird, bevor die Feldbreite erschöpft ist, wird das Feld an diesem Punkt beendet.

Der Buchstabe am Ende gibt die Art der anzuwendenden Umwandlung auf das Argument an. Er kann einer der folgenden sein:

b Das Feld wird als binäre Integer angesehen und für die Ausgabe in einen Integerwert umgewandelt. Das entsprechende Argument sollte eine Integeradresse sein. Führende Nullen werden ignoriert. Diese Spezifikation gibt es nur in Small-C und bei der Anwendung sollte man das berücksichtigen.

c Dieses Feld wird als einzelnes Zeichen ohne Umwandlung akzeptiert. Die Angabe verhindert das normale Auslassen von Leerzeichen. Das Argument für so ein Feld sollte eine Zeichenadresse sein.

d Das Eingabefeld wird als dezimaler Integer (möglicherweise mit Vorzeichen) angesehen und in einen Integerwert umgewandelt. Führende Nullen werden ignoriert.

o Das Feld wird als oktaler Integer angesehen und in einen Integerwert umgewandelt. Das entsprechende Argument sollte eine Integeradresse sein. Führende Nullen werden ignoriert.

s Das Feld sollte als Zeichenkette angesehen werden und mit einer Null am Ende an der Zeichenadresse ihres Argumentes gespeichert werden. An der Adresse muß genug Platz für die Zeichenkette und die Null vorhanden sein. Man sollte daran denken, daß man eine maximale Feldbreite angeben kann, um einen Überlauf zu verhindern. Die Angabe *%1s* wird das nächste Grafikzeichen lesen, wo hingegen *%c* das nächste Zeichen liest, was auch immer es ist.

u Das Argument wird als dezimaler Integer ohne Vorzeichen angesehen und in einen Integerwert umgewandelt. Das entsprechende Argument sollte eine Integeradresse sein. Führende Nullen werden ignoriert. Diese Spezifikation gibt es nur in Small-C und bei der Verwendung sollte man das berücksichtigen.

x Das Feld wird als hexadezimale Zahl angesehen und in einen Integerwert umgewandelt. Das entsprechende Argument sollte eine Integeradresse sein. Führende Nullen oder *0x* oder *0X* werden ignoriert.

Scanf() durchsucht die Kontrollzeichenkette von links nach rechts und verarbeitet Eingabefelder bis die Kontrollzeichenkette erschöpft ist oder ein Feld gefunden wird, das mit den Umwandlungsanweisungen nicht übereinstimmt. Wenn der von *scanf()* zurückgegebene Wert kleiner als die Anzahl

der Umwandlungsanweisungen ist, trat entweder ein Fehler auf oder das Ende der Eingabedatei ist erreicht worden. Wenn keine Felder verarbeitet wurden, wird EOF zurückgegeben, da das Dateiende erreicht wurde.

`fscanf(fd, str, arg1, arg2, ...) int fd; char *str;`

Diese Funktion arbeitet wie *scanf()*, nur wird die Eingabe aus der Datei *fd* genommen.

Funktionen für Formatkonvertierungen

`atoi(str) char *str;`

Diese Funktion wandelt die dezimale Zahl, die durch die Zeichenkette *str* dargestellt wird, in eine Integer um und gibt ihren Wert zurück. Führende Leerzeichen werden übergangen und wahlweise kann der am weitesten links stehenden Ziffer ein Vorzeichen vorausgehen (+ oder-). Das erste nicht-numerische Zeichen beendet die Umwandlung.

`atoib(str, base) char *str; int base;`

Diese Small-C-Funktion wandelt eine Integer ohne Vorzeichen mit Basis *base*, die durch die Zeichenkette *str* dargestellt wird, in einen Integer um und gibt den Wert zurück. Führende Leerzeichen werden übersprungen. Das erste nicht numerische Zeichen beendet die Umwandlung.

`itoa(nbr, str) int nbr; char *str;`

Diese Funktion wandelt eine Zahl in *nbr* in eine dezimale Zeichenkette in *str* um. Das Ergebnis wird in *str* links ausgerichtet, mit führendem Minuszeichen wenn *nbr* negativ ist. Ein Null-Zeichen beendet die Zeichenkette, die groß genug sein muß, um das Ergebnis aufnehmen zu können.

`itoab(nbr, str, base) int nbr; char *str; int base;`

Diese Small-C-Funktion wandelt die Integer *nbr* ohne Vorzeichen in die entsprechende Zeichenkettendarstellung in *str* mit Basis *base* um. Das Ergebnis wird in *str* links ausgerichtet. Ein Null-Zeichen beendet die Zeichenkette, die für das Ergebnis groß genug sein muß.

`dtoi(str, nbr) char *str; int *nbr;`

Diese Funktion wandelt eine dezimale Zahl (eventuell mit Vorzeichen) in der Zeichenkette *str* in eine Integer *nbr* um und gibt die Länge des gefundenen numerischen Feldes zurück. Die Umwandlung wird beendet, wenn das Ende der Zeichenkette oder ein ungültiges Zeichen gefunden wird. Es

werden höchstens ein führendes Vorzeichen und fünf Ziffern benutzt. Bei Werten über 32767 gibt *Dtoi()* ERR zurück.

```
otoi(str, nbr) char *str; int *nbr;
```

Diese Small-C-Funktion wandelt eine oktale Zahl in der Zeichenkette *str* in eine Integer *nbr* um und gibt die Länge des gefundenen oktalen Feldes zurück. Wenn eine nicht-oktale Ziffer in *str* gefunden wird, wird gestoppt. Beim Arbeiten mit 16-Bit-Integern werden höchstens ein führendes Vorzeichen und fünf Ziffern benutzt. Bei einer oktalen Ziffer größer als 177777 gibt *atoi()* ERR zurück.

```
utoi(str, nbr) char *str; int *nbr;
```

Diese Small-C-Funktion wandelt eine dezimale Zahl ohne Vorzeichen, dargestellt durch die Zeichenkette *str*, in eine Integer *nbr* um und gibt die Länge des gefundenen numerischen Feldes zurück. Gestoppt wird, wenn das Ende der Zeichenkette erreicht wird oder wenn ein nicht dezimales Zeichen gefunden wird. Beim Arbeiten mit 16-Bit-Integern werden höchstens ein führendes Vorzeichen und fünf Ziffern benutzt. Bei einer Zahl größer als 65535 gibt *utoi()* ERR zurück.

```
xtoi(str, nbr) char *str; int *nbr;
```

Diese Small-C-Funktion wandelt eine hexadezimale Zahl in der Zeichenkette *str* in ein Integer *nbr* um und gibt die Länge des gefundenen hexadezimalen Feldes zurück. Gestoppt wird, wenn das Ende der Zeichenkette erreicht wird oder wenn ein nicht-hexadezimales Zeichen gefunden wird. Beim Arbeiten mit 16-Bit-Integern werden höchstens ein führendes Vorzeichen und fünf Ziffern benutzt. Wenn mehr Ziffern vorhanden sind, gibt *xtoi()* ERR zurück.

```
itod(nbr, str, sz) int nbr, sz; char *str;
```

Diese Small-C-Funktion wandelt *nbr* in eine Zahl mit Vorzeichen (wenn negativ) in *str* um. Das Ergebnis erscheint rechtsbündig und mit Leerzeichen gefüllt in *str*. Das Vorzeichen und mögliche höherwertige Ziffern werden abgeschnitten, wenn die Zielzeichenkette zu klein ist. Zurückgegeben wird *str*. *Sz* zeigt die Länge der Zeichenkette an. Wenn *sz* größer Null ist, erscheint ein Null-Byte bei *str[sz-1]*. Wenn *sz* Null ist, zeigt eine Suche nach dem ersten *str* folgenden Null-Byte das Ende der Zeichenkette an. Wenn *sz* kleiner Null ist, werden alle *sz* Zeichen von *str* benutzt, einschließlich dem letzten.

```
itoo(nbr, str, sz) int nbr, sz; char *str;
```

Diese Small-C-Funktion wandelt *nbr* in eine oktale Zeichenkette in *str* um. Das Ergebnis erscheint rechtsbündig und mit Leerzeichen gefüllt in der Zielzeichenkette. Höherwertige Ziffern werden abgeschnitten, wenn die Zielzeichenkette zu klein ist. Zurückgegeben wird *str*. *Sz* zeigt die Länge der Zeichenkette an. Wenn *sz* größer Null ist, erscheint ein Null-Byte bei *str[sz-1]*. Wenn *sz* Null ist, zeigt eine Suche nach dem ersten *str* folgenden Null-Byte das Ende der Zeichenkette an. Wenn *sz* kleiner Null ist, werden alle *sz* Zeichen von *str* benutzt, einschließlich dem letzten.

```
itou(nbr, str, sz) int nbr, sz; char *str;
```

Diese Small-C-Funktion wandelt *nbr* in eine dezimale Zeichenkette ohne Vorzeichen in *str* um. Sie arbeitet ähnlich *itod()*, nur daß das höherwertige Bit von *nbr* als Vorzeichenbit angesehen wird.

```
itox(nbr, str, sz) int nbr, sz; char *str;
```

Diese Small-C-Funktion wandelt *nbr* in eine hexdezimale Zeichenkette in *str* um. Das Ergebnis erscheint rechtsbündig und mit Leerzeichen gefüllt in der Zielzeichenkette. Höherwertige Ziffern werden abgeschnitten, wenn die Zielzeichenkette zu klein ist. Zurückgegeben wird *str*. *Sz* zeigt die Länge der Zeichenkette an. Wenn *sz* größer Null ist, erscheint ein Null-Byte bei *str[sz-1]*. Wenn *sz* Null ist, zeigt eine Suche nach dem ersten *str* folgenden Null-Byte das Ende der Zeichenkette an. Wenn *sz* kleiner Null ist, werden alle *sz* Zeichen von *str* benutzt, einschließlich dem letzten.

Funktionen für die Zeichenkettenbehandlung

```
left(str) char *str;
```

Diese Small-C-Funktion richtet eine Zeichenkette in *str* linksbündig aus. Beginnend mit dem ersten nicht-leeren Zeichen und fortfahrend bis zum Null-Byte am Schluß wird die Zeichenkette an die durch *str* angegebene Adresse gebracht.

```
pad(str, ch, n) char *str, ch; int n;
```

Diese Small-C Funktion füllt die Zeichenkette bei *str* n-Mal mit dem Zeichen *ch*.

```
reverse(str) char *str;
```

Diese Funktion kehrt die Reihenfolge der Zeichen in der mit Null abgeschlossenen Zeichenkette in *str* um.

```
strcat(dest, sour) char *dest, *sour;
```

Diese Funktion fügt die Zeichenkette in *sour* ans Ende der Zeichenkette *dest* an. Das Null-Zeichen am Ende von *dest* wird durch das erste Zeichen von *sour* ersetzt. Ein Null-Zeichen beendet die neue *dest*-Zeichenkette. Der für *dest* reservierte Platz muß groß genug sein, um das Ergebnis aufnehmen zu können. Diese Funktion gibt *dest* zurück.

```
strncat(dest, sour, n) char *dest, *sour; int n;
```

Diese Funktion arbeitet ähnlich *strcat()*, nur das maximal *n* Zeichen von der Quellzeichenkette zur Zielzeichenkette übertragen werden.

```
strcmp(str1, str2) char *str1, *str2;
```

Diese Funktion gibt eine Integer kleiner als, gleich oder größer als Null zurück, abhängig davon, ob die Zeichenkette *str1* kleiner als, gleich oder größer als die Zeichenkette *str2* ist. Die Zeichen werden einzeln miteinander verglichen, beginnend am linken Ende der Zeichenketten bis ein Unterschied gefunden wird. Der Vergleich basiert auf dem numerischen Wert der Zeichen. *Str2* hat eine geringere Wertigkeit als *str1*, wenn *str2* gleich aber kürzer als *str1* ist und umgekehrt.

```
lexcmp(str1, str2) char *str1, *str2;
```

Diese Small-C-Funktion arbeitet wie *strcmp()*, nur das ein lexikografischer Vergleich benutzt wird. Damit die Ergebnisse sinnvoll sind, sollten nur ASCII-Zeichen (0-127 dezimal) in den Zeichenketten erscheinen. Buchstaben werden in Wörterbuchreihenfolge verglichen, wobei Großbuchstaben gleich den Kleinbuchstaben sind. Sonderzeichen gehen den Buchstaben voraus, wobei den Sonderzeichen wiederum die Kontrollzeichen vorausgehen, mit Ausnahme von DEL, das den höchsten Wert hat.

```
strncmp(str1, str2, n) char *str1, *str2; int n;
```

Diese Funktion arbeitet ähnlich *strcmp()*, nur das maximal *n* Zeichen verglichen werden.

```
strcpy(dest, sour) char *dest, *sour;
```

Diese Funktion kopiert die Zeichenkette bei *sour* nach *dest*. *Dest* wird zurückgegeben. *Dest* muß für das Ergebnis groß genug sein.

```
strncpy(dest, sour, n) char *dest, *sour; int n;
```

Diese Funktion arbeitet wie *strcpy()*, nur daß *n* Zeichen in die Zielzeichenkette gebracht werden, gleichgültig wie lang die Quellzeichenkette

ist. Wenn die Quellzeichenkette zu kurz ist, wird mit Nullen aufgefüllt. Wenn sie zu lang ist, wird sie bei *dest* abgeschnitten. Ein Null-Zeichen folgt dem letzten in die Zielzeichenkette gebrachten Zeichen.

`strlen(str) char *str;`

Diese Funktion gibt die Anzahl der Zeichen in der Zeichenkette *str* zurück. Das Null-Zeichen am Ende der Zeichenkette wird nicht gezählt.

`strchr(str, c) char *str, c;`

Diese Funktion gibt einen Zeiger auf das erste Vorkommen des Zeichens *c* in der Zeichenkette *str* zurück. Sie gibt NULL zurück, wenn das Zeichen nicht gefunden wird. Die Suche wird mit dem ersten Null-Zeichen beendet.

`strrchr(str, c) char *str, c;`

Diese Funktion arbeitet wie *strchr()*, nur daß das am weitesten rechts stehende Vorkommen des Zeichens gesucht wird.

Funktionen für die Zeichenklassifizierung

Die folgenden Funktionen entscheiden, ob ein Zeichen zu einer bestimmten Zeichenklasse gehört. Sie geben bei Übereinstimmung "true" (nicht Null) zurück und "false" (null) bei keiner Übereinstimmung.

`isalnum(c) char c;`

Diese Funktion stellt fest, ob *c* alphanumerisch (A-Z, a-z, oder 0-9) ist.

`isalpha(c) char c;`

Diese Funktion stellt fest, ob *c* ein Buchstabe (A-Z oder a-z) ist.

`isascii(c) char c;`

Diese Funktion stellt fest, ob *c* ein ASCII-Zeichen (dezimale Werte 0-127) ist.

`iscntrl(c) char c;`

Diese Funktion stellt fest, ob *c* ein Kontrollzeichen (ASCII-Code 0-31 oder 127) ist.

`isdigit(c) char c;`

Diese Funktion stellt fest, ob *c* eine Ziffer (0-9) ist.

`isgraph(c) char c;`

Diese Funktion stellt fest, ob *c* ein grafisches Symbol (ASCII-Codes 33-126) ist.

`islower(c) char c;`

Diese Funktion stellt fest, ob *c* ein Kleinbuchstabe (ASCII-Codes 97-122) ist.

`isprint(c) char c;`

Diese Funktion stellt fest, ob *c* ein druckbares Zeichen (ASCII-Codes 32-126) ist. Leerzeichen werden als druckbar betrachtet.

`ispunct(c) char c;`

Diese Funktion stellt fest, ob *c* ein Interpunktionszeichen (alle ASCII-Codes außer Kontrollzeichen und alphanumerische Zeichen) ist.

`isspace(c) char c;`

Diese Funktion stellt fest, ob *c* ein Leerstellenzeichen (ASCII SP, HT, VT, CR, LF oder FF) ist.

`isupper(c) char c;`

Diese Funktion stellt fest, ob *c* ein Großbuchstabe ist (ASCII-Codes 65-90).

`isxdigit(c) char c;`

Diese Funktion stellt fest, ob *c* ein hexadezimales Zeichen ist (0-9, A-F oder a-f).

`lexorder(c1, c2) char c1, c2;`

Diese Small-C-Funktion gibt eine Integer kleiner als, gleich oder größer als Null zurück, abhängig davon, ob *c1* lexikographisch kleiner als, gleich oder größer als *c2* ist. Damit die Ergebnisse sinnvoll sind, sollten nur ASCII-Zeichen (0-127 dezimal) übergeben werden. Buchstaben werden in Wörterbuchreihenfolge verglichen, wobei Großbuchstaben gleich den Kleinbuchstaben sind. Sonderzeichen gehen den Buchstaben voraus, wobei den Sonderzeichen wiederum die Kontrollzeichen vorausgehen, mit Ausnahme von DEL, das den höchsten Wert hat.

Funktionen für die Zeichenumwandlung

`toascii(c) char c;`

Diese Funktion gibt das ASCII-Äquivalent von *c* zurück. Bei Systemen, die den ASCII-Zeichensatz benutzen, wird *c* einfach unverändert zurückgegeben. Diese Funktion ermöglicht es die Eigenheiten des ASCII-Codes zu benutzen ohne Implementationsabhängigkeiten in ein Programm zu bringen.

`tolower(c) char c;`

Diese Funktion gibt den Kleinbuchstaben *c* zurück, wenn *c* ein Großbuchstabe ist; sonst wird *c* unverändert zurückgegeben.

`toupper(c) char c;`

Diese Funktion gibt den Großbuchstaben *c* zurück, wenn *c* ein Kleinbuchstabe ist; sonst wird *c* unverändert zurückgegeben.

Mathematische Funktionen

`abs(nbr) int nbr;`

Diese Funktion gibt den Absolutwert von *nbr* zurück.

`sign(nbr) int nbr;`

Diese Funktion gibt -1, 0 oder +1 zurück, abhängig davon ob *nbr* kleiner als, gleich oder größer als Null ist.

Funktionen für die Programmkontrolle

`calloc(nbr, sz) int nbr, sz;`

Diese Funktion ordnet *nbr**sz* Bytes auf Null gesetzten Speicher zu. Bei Erfolg wird die Adresse des Speicherblocks zurückgegeben, sonst Null.

`malloc(nbr) int nbr;`

Diese Funktion ordnet *nbr* Bytes uninitialisierten Speicher zu. Bei Erfolg wird die Adresse des Speicherblocks zurückgegeben, sonst Null.

`avail(abort) int abort;`

Diese Small-C-Funktion gibt den freien Speicherplatz zwischen dem Programm und dem Stapel zurück. Geprüft wird auch, ob sich der Stapel mit dem zugeordneten Speicher überlappt; wenn ja und wenn *abort* nicht Null ist, wird das Programm abgebrochen und auf dem Bildschirm erscheint der Buchstabe S, der einen Stapelfehler anzeigt. Wenn jedoch *abort* Null ist,

gibt *avail()* Null an den Aufrufer zurück. Durch diese Funktion kann man den vollen Speicherplatz ausnutzen. Man sollt jedoch darauf achten, daß für die Benutzung des Stapels genug Platz bleibt.

`free(addr) char *addr; (alias cfree)`

Diese Funktion gibt ab Adresse *addr* einen Block zugeordneten Speichers frei. Bei Erfolg wird *addr* zurückgegeben, sonst NULL. Es ist notwendig den Speicher in umgekehrter Reihenfolge wieder freizugeben wie er zugeordnet worden war. Man sollte vermeiden Speicher freizugeben bevor eine Datei geöffnet wurde, da die *open*-Funktion dynamisch Puffer und FCB-Platz zuordnet. Man darf nicht davon ausgehen, daß durch Schließen einer Datei ihr Speicher freigegeben wird

```
getarg(nbr,str,sz,argc,argv)
char *str; int nbr, sz, argc, *argv;
```

Diese Small-C-Funktion findet das Argument aus der Befehlszeile, das durch *nbr* angegeben wird, bringt es (mit Null abgeschlossen) in die Zeichenkette *str* mit maximaler Länge *sz* und gibt die Länge des erhaltenen Feldes zurück. *Argc* und *argv* müssen für die Funktion *main()* die gleichen Werte zur Verfügung stellen, wenn das Programm gestartet wird. Wenn *nbr* Null ist, ist der Programmname gewünscht. Wenn *nbr* 1 ist, ist das erste Argument nach dem Programmnamen erwünscht und so weiter. CP/M gibt den Programmnamen an ein Programm nicht weiter, daher wird an diese Stelle ein Stern gesetzt. Wenn kein Argument *nbr* entspricht, bringt *getarg()* ein Nullbyte nach *str* und gibt EOF zurück.

`poll(pause) int pause;`

Diese Small-C-Funktion fragt die Tastatur nach einer Eingabe des Benutzer ab. Wenn keine Eingabe ansteht, wird Null zurückgegeben. Wenn ein Zeichen wartet, bestimmt der Wert von *value*, was geschieht. Wenn *pause* Null ist, wird das Zeichen sofort zurückgegeben. Wenn *pause* nicht Null ist und das Zeichen ein Control-S ist, gibt es eine Pause in der Programmausführung; wenn das nächste Zeichen über die Tastatur eingegeben wird, wird Null an den Aufrufer übergeben. Wenn das Zeichen Control-C ist, wird die Programmausführung beendet. Alle anderen Zeichen werden sofort an den Aufrufer zurückgegeben.

`exit(errcode) int errcode; (alias abort)`

Diese Funktion schließt alle offenen Dateien und springt ins Betriebssystem zurück. Wenn *errcode* ungleich Null ist, wird der Code zum Bildschirm geschrieben; ein Programm, das mit der Anweisung *exit(7)* (Code 7 = Control-G, Piepser) endet, läßt den Lautsprecher ertönen.

4 Das Small-Mac-Assemblerpaket

Small-Mac ist ein Makro-Assembler für 8080-/Z80-Systeme unter CP/M. der für den Small-C-Compiler entwickelt wurde. Als solcher soll er die Small-C-Benutzer durch seine betonte Einfachheit, Portabilität, Adaptierbarkeit und seinen Lehreffekt ansprechen. Diesen primären Zielen wurden Programmgröße und Ausführungszeit untergeordnet. Daher wurde Small-Mac, genau wie der Small-C-Compiler, in Small-C geschrieben und wird im Quellcode und Objektcode geliefert. Die hervorstechendsten Eigenschaften des Small-Mac-Pakets sind:

- einfach zu handhaben
- Makro-Möglichkeiten
- Operatoren für Ausdrücke aus der C-Sprache
- anschauliche Fehlermeldungen
- extern definierte Maschineninstruktionstabelle

Folgende Programme sind im Small-Mac-Paket enthalten:

MAC	Makroassembler
LNK	Linker
LGO	Lader (laden-und-ausführen, load and go)
LIB	Bibliotheksverwalter
CMIT	CPU-Anpassungsprogramm
DREL	Dump relokatierbarer Objektdateien

MAC ist ein tabellengesteuerter Makroassembler mit zwei Läufen, der verschiebbaren (relokatierbaren) Code erzeugt. Er lernt die Zielmaschine aus einer Maschineninstruktionstabelle (MIT), die mit einem Texteditor erzeugt und mit der Anpassungsutility CMIT kompiliert wird. Small-Mac erzeugt relokatierbare Objektmodule im 8-Bit-Microsoft-Format und wird durch eine einfache Befehlssyntax aufgerufen.

LNK ist der Linker für Small-Mac. Er kombiniert Objektmodule mit Modulen aus einer Bibliothek zu einem vollständigen, ausführbaren Programm. Die Standardausgabe ist eine COM-Datei. Er kann eine Laden-und-Ausführen Datei (LGO, load-and-go) für die Ausführung an einer gewünschten Adresse erzeugen.

LGO lädt und führt wahlweise LGO-Dateien aus. Er ist äußerst nützlich, um Systemerweiterungen nach dem Booten des Betriebssystem zu laden.

LIB erstellt, wartet und listet den Inhalt von LNK-kompatiblen Bibliotheken. Dadurch kann man Module assemblieren, die verschiedenen Programmen gemeinsam sind, sie in einer einzigen Bibliothek zusammenfassen und dann LNK die Bibliothek nach den Modulen suchen lassen, die von dem Programm gebraucht werden, das gelinkt werden soll.

CMIT kompiliert Tabellen für Maschineninstruktionen, listet sie und konfiguriert den Assembler wahlweise mit der sich ergebenden Objekttabelle. Dieser Definitionsansatz für die Assemblermaschineninstruktion ist sehr flexibel, und erlaubt es, den Assembler an verschiedene CPUs anzupassen und spezielle Maschineninstruktionen ohne den gesamten Überbau der Makroverarbeitung anzulegen.

DREL erzeugt eine formatierte Hex-Ausgabe (Dump) von REL- und LIB-Dateien. So kann man den Inhalt dieser Dateien zu studieren, obwohl sie auf Bitebene strukturiert sind.

Systemanforderungen

Diese Implementation des Small-Mac-Paketes läuft auf 8080-/8085-/Z80-Systemen, die das CP/M-Betriebssystem verwenden. Man sollte ein Diskettenlaufwerk und mindestens 56 KByte Speicher haben.

Quelldateien

Quellzeilen haben ein freies Format, mit Feldern, die in der Zeile in der folgenden Reihenfolge erscheinen:

```
Symbol/Label   Operation   Operand   Kommentar
```

Jedes Feld ist optional und Leerzeilen werden ignoriert. Felder werden durch Leerstellen (Leerzeichen und Tabs) getrennt; Kommentaren geht ein Semikolon voraus (;).

Das Feld Symbol/Label

Ein Symbol besteht aus einer zusammenhängenden Sequenz von Buchstaben, Ziffern und aus den Sonderzeichen _ . $? oder @. Das erste Zeichen darf keine Ziffer sein. Klein- und Großbuchstaben sind gleichbedeutend. Symbole können beliebig lang sein, aber nur die ersten 8 Zeichen sind signifikant und nur sechs Zeichen werden für die Deklaration externer Referenzen und Einsprungspunkte in Objektmodulen benutzt. Label im Symbol/Label-Feld werden immer mit einem Doppelpunkt abgeschlossen (:). Sie können allein oder gefolgt von einer Anweisung und/oder einem Kommentar erscheinen. Die Adresse, die dem Label zugewiesen wird, ist die Adresse des ersten Bytes der nächsten Instruktion oder Datenbereichs.

Zwei Doppelpunkte, die einem Label folgen, definieren es als Einsprungspunkt. Verweise auf Label und Symbole dürfen nicht durch Doppelpunkt beendet werden.

Das Operationsfeld

Wenn dem Operationscode (Opcode) oder den Assembleranweisungen (Pseudo-Op) keine Label oder Symbole vorausgehen, kann das Operationsfeld in der ersten Zeichenstelle beginnen. Die Assembleranweisungen werden innerhalb des Assemblers definiert, die Opcodes, die Maschineninstruktionen, werden jedoch in einer externen Maschineninstruktionstabelle definiert, die durch das Anpassungsprogramm CMIT in das interne Format kompiliert und in den Assembler gebracht wird.

Das Operandenfeld

Operanden und/oder Operandenstellen (Speicherstellen oder Registernamen) werden im Operandenfeld angegeben.

Symbole im Operandenfeld müssen entweder irgendwo in der Datei definiert oder als extern deklariert werden. Dies erreicht man, indem man sie entweder mit zwei ##-Zeichen abschließt oder indem man die Anweisung EXT benutzt.

Das Dollarzeichen ($) kann im Operandenfeld als Label für die aktuelle Adresse der Anweisung benutzt werden.

Speicherverweise und numerische Werte können als Ausdrücke geschrieben werden. Die Operatoren in Small-Mac-Ausdrücken sind ein Subset von denen der C-Sprache und folgen den gleichen Vorrang- und Anordnungsregeln. Damit braucht der C-/Assembler-Programmier nur einmal die Regeln zu lernen. Mehr über Ausdrücke erscheint unten im Abschnitt "Ausdrücke".

Kommentare

Kommentare können als letztes Feld in einer Zeile erscheinen. Ein Semikolon zeigt den Beginn von Kommentaren an. Eine Zeile kann vollständig aus einem Kommentar bestehen, wenn das Semikolon als erstes Zeichen angegeben wird. Leerzeichen vor Kommentaren sind nicht nötig.

Objektdateien

Small-Mac kennt zwei Arten von Objektdateien: Module und Bibliotheken. Module werden durch den Assembler angelegt und haben immer die Datei-

namenserweiterung REL. Bibliotheken werden durch den Bibliotheksmanager erzeugt und bestehen aus einem Dateipaar: der Bibliothek selbst (Erweiterung LIB) und einem Index für die Bibliothek (Erweiterung NDX)

Die Small-Mac-Objektmoule sind verschiebbar und folgen dem 8-Bit-Microsoft-Format. Dies ist ein Bitfeldformat, um die Größe von Objektdateien zu reduzieren. Es gibt keine Byteausrichtung, außer am Beginn eines Moduls. Die Bitfelder sind in der folgenden Tabelle aufgeführt:

LINK FORMAT	BESCHREIBUNG	BENUTZT?
0 <8-Bits>	Absolutes Feld	JA
1000000 lll <Zeiket>	Eingangssymbol	JA
1000001 lll <Zeiket>	allg. Block wählen	NEIN
1000010 lll <Zeiket>	Programmname	JA
1000011 lll <Zeiket>	zu durchsuch. Bibliothek	NEIN
1000100 lll <Zeiket>	erweitertes Link-Feld	NEIN
1000101 tt <16-Bits> lll <Zeiket>	allg. Bereichsgröße	NEIN
1000110 tt <16-Bits> lll <Zeiket>	externe Verweiskette	JA
1000111 tt <16-Bits> lll <Zeiket>	Einsprungspunkt	JA
1001000 tt <16-Bits>	externen Verweis dekre.	NEIN
1001001 tt <16-Bits>	externen Verweis inkre.	JA
1001010 tt <16-Bits>	Datenbereichsgröße	NEIN
1001011 tt <16-Bits>	Positionszähler setzen	JA
1001100 tt <16-Bits>	Positionszählerkette	NEIN
1001101 tt <16-Bits>	Programmbereichsgröße	JA
1001110 tt <16-Bits>	Modulende	JA
1001111	Dateiende	JA
101 <16-Bits>	Programmrelatives Feld	JA
110 <16-Bits>	Datenrelatives Feld	NEIN
111 <16-Bits>	Allgemein relatives Feld	NEIN

Das Feld lll belegt 3 Bits und gibt die Zeichenkettenlänge an. Das Feld tt belegt 2 Bits und gibt den Typ des folgenden 16-Bit Feldes an. Die Typcodes sind:

CODE	BEDEUTUNG	BENUTZT?
00	absolut	JA
01	programmrelativ	JA
10	datenrelativ	NEIN
11	allgemein relativ	NEIN

Die Reihenfolge in einem Modul lautet:

1. Programmname
2. Eingangssymbole (ungeordnet)
3. Programmbereichsgröße
4. Eigentliches Modul
   absoluter Teil
   programmrelativer Teil
   externe Referenz Inkrement
   Teile für den Positionszähler setzen
5. Liste (in alphanumerischer Reihenfolge) von
   externen Referenzketten
   Einsprungspunkten
6. Programmende
7. Dateiende

Eine Bibliothek ist eine Verkettung von Modulen, mit dem einen Unterschied, daß es nur eine Dateiendekennung gibt. Ein Bibliotheksindex sind bloße Wortpaare, die auf den Anfang eines jeden Moduls in der Bibliothek zeigen. Das erste Wort gibt den 128-Byte-Block an, das zweite das Byte im Block. Beide beginnen bei Null.

Programmrelative Felder können Adressen, Daten und Zeiger auf Ketten sein. Das Ende einer Kette wird durch ein absolutes Feld mit dem Wert Null angezeigt. Alle externen Referenzen zu einem Einsprungspunkt sind mit der Adresse des entsprechenden Einsprungspunktes verkettet, so daß der Linker sie finden unsd ersetzen (auflösen) kann. Wenn eine externe Referenz in eine Anweisung eingeschlossen ist, so daß das Ergebnis ein Offset (negativ oder positiv) von der Referenz ist, dann geht der Kette unmittelbar ein externes Referenzinkrement voraus. Dieser wird nach der Auflösung dazu addiert. Das Inkrementfeld hat keinen Effekt auf den zu ladenden Positionszähler. Small-Mac-Assemblerlistings zeigen sowohl den Zeiger auf die Kette als auch den Inkrementwert.

Maschineninstruktionen

Wie oben schon erwähnt, werden Maschineninstruktionen in einer externen Maschineninstruktionstabelle definiert (MIT). Dies ist eine ASCII-Datei, in der die Operationmnemoniks, die Syntax der Operanden und der Objektcode für jede Maschineninstruktion enthalten ist. Das Anpassungsprogramm CMIT kompiliert diese Tabelle in internes Format, listet sie dann wahlweise und/oder fügt eine Kopie in den Assembler ein.

Dieser Ansatz der Definition der Maschineninstruktionstabelle macht es sehr viel einfacher, den Assembler auch an andere CPUs adaptieren. Weiter

wird dadurch auch das Anlegen besonderer Instruktionssätze vereinfacht, die sofort assembliert werden können, ohne den Aufwand für eine Makroverarbeitung.

In Anhang D sind die Maschineninstruktionstabellen für die 8080- und die Z80-Prozessoren abgedruckt. Jede Zeile besteht aus drei Feldern: Objekt, Menmonic und Operand. Leerstellen trennen die Felder.

Ausdrucksbeschreiber können in den Operand- und Objektfelder erscheinen. Sie zeigen an, wo in der Operandensystax eine Ausdruck erscheinen kann und wo dann im Objektcode der entsprechende Ausdruckswert zu stehen hat. Ebenso wird auch die Größe des Objektwertes angegeben (ein oder zwei Byte) und ob dies ein zum Programmzähler (PC) relativer Wert ist oder nicht. Ausdrucksbeschreiber bestehen aus den Kleinbuchstaben x oder p und (im Objektfeld) einer Ziffer, die die Anzahl der Bytes angibt (ein oder zwei). Der Buchstabe x kennzeichnet einen normalen Ausdruckswert und der Buchstabe p einen PC-relativen Wert. Außer für diese Beschreibungen müssen alle anderen Bezeichnungen in Großbuchstaben erscheinen.

In der kompilierten MIT werden 16 Bits benutzt, um das Format des Objektcodes zu beschreiben. Jedes Codebyte benutzt ein Bit im Formatwort und jeder Ausdruck benutzt drei Bits. Daher kann jede Kombination von Bytes und Ausdrücken erzeugt werden, solange die Zahl der benutzten Formatbits 16 nicht übersteigt.

Zwischen den Objektkomponenten werden Unterstriche für die bessere Lesbarkeit verwendet. Die Codebytes und die Ausdruckswerte werden in der im Objektfeld angegeben Reihenfolge erzeugt. Wenn mehr als ein Ausdruck angegeben wird, stellt sie der Assembler genau in der Reihenfolge in die Objektdatei, wie sie im Operandenfeld erscheinen.

Damit eine Instruktion mit einer Eintragung in der MIT übereinstimmt, führt der Assembler für ein Mnemonik eine Hashsuche durch und dann eine serielle Suche für die korrekte Operandenvariante. Die serielle Suche verläuft in der Reihenfolge der Varianten im MIT. Dies geht gut bei 8080-Assemblermnemoniks aber ziemlich schlecht beim Z80, der für einige Mnemoniks eine ganze Menge Varianten hat (zum Beispiel LD). Wenn man etwas über die relative Häufigkeit der benutzten Operandenvarianten weiß, kann man die Reihenfolge in der MI-Tabelle ändern, um Suchzeit zu sparen. Alle Varianten eines gegebenen Mnemoniks müssen jedoch zusammenbleiben.

Durch die Benutzung von senkrechten Strichen im Operandenfeld der MIT kann man Platz sparen, wegen der Redundanz der Operandenvarianten in

den meisten Instruktionssätzen und der Tendenz der CP/M-Architekten, Objektcodes sequentiell zuzuordnen. Durch den senkrechten Strich werden die Varianten auf der gleichen Zeile voneinander getrennt. In solchen Fällen wird das Objektbyte, das dem ersten Operanden unmittelbar vorausgeht oder zuletzt kommt, wenn es keinen Ausdruck gibt, auf die erste Variante angewendet. Für jede folgende Variante auf der gleichen Zeile wird dieses Byte um eins erhöht. Die anderen Objektbytes bleiben unverändert.

Wenn man plant, die MIT zu ändern, ist es von äußerster Wichtigkeit, daß man verstanden hat, wie der Assembler die Übereinstimmung zwischen einem Ausdruck und einem Ausdrucksbeschreiber aus dem Operandenfeld im MIT findet. Für ein besseres Verständnis sollte man sich die Datei MIT.C ansehen. Wenn einmal das Instruktionsmnemonik im MIT gefunden wurde, wird *match()* aufgerufen, um zu versuchen eine Übereinstimmung mit der Operandenvariante zu finden. Sie vergleicht Zeichen von links nach rechts, wobei Groß- und Kleinbuchstaben gleich behandelt werden. Wenn keine Übereinstimmung gefunden wird, scheitert diese Variante; bei Erfolg wird das nächste Zeichenpaar verglichen. Wenn im MIT-Operandenfeld ein *p* oder *x* gefunden wird, wird die Zeichenkette, die mit dem aktuellen Instruktionszeichen beginnt, ausgelassen, bis ein Komma oder eine unpaarige rechte Klammer erscheint. Unpaarig heißt in diesem Fall, nicht gefunden während des Überlesens der Ausdruckzeichenkette.

Wenn zum Beispiel die Instruktion "LD A,((a+b)/2)" als übereinstimmend mit "LD A,(x)" gefunden wird, dann beendet die zweite rechte Klammer das Überlesen der Ausdrücke, weil nur die erste Klammer im Überleseprozeß gefunden wird. Also wird die Zeichenkette "(a+b)/2" als Ausdruck interpretiert. Dieser Teil der Instruktion wird aus der Quellzeile während des Suchverfahrens herausgelöst und in einen separaten Puffer gebracht, zur weiteren Analyse.

Was würde nun passieren, wenn die Anweisung "LD A,(x)" im MIT vor "LD A,(HL)" erscheinen würde, während die Instruktion "LD A,(HL)" gerade assembliert würde? Richtig, HL würde als Ausdruck interpretiert und die Instruktion würde irrtümlichweise als "LD A,(x)" gefunden werden. Daher sollte man vorsichtig sein, wenn solche Varianten nach anderen Varianten kommen, die übereinstimmen könnten. Übrigens würde dieser Fehler ohne Zweifel eine Assemblerfehlermeldung produzieren.

Assembleranweisungen

Small-Mac unterstützt die Assembleranweisungen (Pseudo-Ops) in der folgenden Tabelle. Diese Small-Mac-Version sieht keine wiederholten Assembleranweisungen oder bedingte Assemblierung vor.

SYNTAX	FUNKTION
[Label] DW Wert[,Wert[,...]]	definiert Worte
[Label] DB Wert[,Wert[,...]]	definiert Bytes
[Label] DS Ausdruck	reserviert Speicher
[Label] EXT Symbol[,Symbol[,...]]	deklariert externe Referenzen
Symbol SET Ausdruck	setzt Symbol auf den Wert von Ausdruck
Symbol EQU Ausdruck	Symbol ist gleich Ausdruck
[Label] ORG Ausdruck	Positionszähler auf Ausdruck setzen
[Label] END [Ausdruck]	Ende der Quelldatei (Ausdruck gibt die Startadresse an)
Symbol MACRO	Beginn einer Makrodefinition
ENDM	Ende einer Makrodefinition
[Label] Makroname [par[,par[,...]]]	Aufrufen (erweitern) des benannten Makros

Bei den Pseudo-Ops aus Tabelle 2 zeigen eckige Klammern optionale Elemente an. Der Term *Wert* steht entweder für einen Ausdruck oder eine Zeichenkette. Zeichenketten werden in Anführungszeichen (") oder Hochkammas (') eingeschlossen. Wenn diese Zeichen auch innerhalb der Zeichenkette erscheinen sollen, müssen zwei aufeinanderfolgende Zeichen erscheinen. Der Term *Ausdruck* steht für einen Ausdruck.

[Label] DW Wert[,Wert[,...]]

Definiert Worte: Für jeden Wert im Operandenfeld reserviert der Assembler ein Wort, das den Wert enthält. Wenn ein Label angegeben ist, wird die Adresse des ersten Wortes angenommen.

[Label] DB Wert[,Wert[,...]]

Definiert Bytes: Für jeden Wert im Operandenfeld reserviert der Assembler ein Byte, das den Wert enthält. Jeder Wert muß absolut sein. Wenn ein Label angegeben ist, wird die Adresse des ersten Bytes angenommen.

[Label] DS Ausdruck

Reserviert Speicherplatz: Die Anzahl der im Ausdruck angegeben Bytes wird reserviert. Sie haben keinen vorhersagbaren Wert. Der Ausdruck muß einen absoluten Wert ergeben. Wenn ein Label angegeben ist, wird die Adresse des ersten Bytes angenommen.

[Label] EXT Symbol[,Symbol[,...]]

Deklariert externe Referenzen: Jedes angegebene Symbol wird als extern deklariert. Wenn ein Label angegeben ist, wird die Adresse der nächsten Anweisung oder des nächsten Datenbytes im Programm angenommen. Ein Symbol als extern zu deklarieren ist ausreichend, damit das Modul, daß es als Einsprungspunkt enthält, durch den Linker zu dem Programm geladen wird. Es muß nicht ausdrücklich darauf Bezug genommen werden.

`symbol SET Ausdruck`

Symbolwert setzen: Dieser Pseudo-Befehl setzt das Symbol auf den Wert des Audrucks. Es kann später durch andere SETs zurückgesetzt werden. Wenn einem Symbol einmal ein Wert zugewiesen wurde, kann es in Ausdrücken benutzt werden.

`symbol EQU expr`

Symbol einem Wert gleichsetzen: Dieser Pseudo-Befehl weist einem Symbol den Wert eines Ausdrucks zu. Dasselbe Symbol kann nicht mehr mit SET oder EQU gesetzt werden. Wenn einmal der Wert einem Symbol zugewiesen wurde, kann er in Ausdrücken benutzt werden.

`[label] ORG Ausdruck`

Anfangswert für Programmzähler einstellen: Dieser Pseudo-Befehl stellt den Wert des Programmzählers ein. In Small-Mac kann der Zähler nur vorwärts bewegt werden. Das hindert Programmierer daran, alten mit neuem Code zu überschreiben und schützt den Linker vor Irrtümern, wenn er versucht, externe Referenzketten aufzulösen.

`[label] END [Ausdruck]`

Ende der Quelldatei: Dieser Pseudo-Befehl bestimmt das Ende der Quelldatei. Er ist erforderlich und muß die letzte Zeile im Programm sein. Wenn ein Label vorhanden ist, wird die programmrelative Adresse des Bytes, das dem letzten assemblierten Byte folgt, unterstellt. Wenn ein Ausdruck vorhanden ist, muß ein programmrelativer Wert erzeugt werden, den der Assembler als Anfangsadresse des Programms nimmt. Es sollte nur eine Startadresse angegeben werden, wenn ein Programm aus mehreren Quelldateien assembliert wird. Wenn jedoch mehrere vorhanden sind, wird der Assembler das zuletzt verarbeitete nehmen. Wenn keine Startadresse angegeben wird, beginnt die Programmausführung mit der ersten Programmanweisung. Die Startadresse, sofern vorhanden, wird in die Ausgabe der Objektdatei eingeschlossen. Wenn der Linker aus mehreren Modulen ein ausführbares Programm zusammenbaut, wird am Programmanfang ein Sprung zur Startadresse eingefügt. Wenn mehr als ein Modul mit einer Startadresse gefunden wird, wird die letzte benutzt.

`symbol MACRO`

Beginn einer Makrodefinition: Dieser Pseudo-Befehl signalisiert den Beginn einer Makrodefinition. Das Symbol ist erforderlich und gibt dem Makro seinen Namen. Mehr über Makros steht unten in "Die Makromöglichkeiten".

ENDM

Ende einer Makrodefinition: Dieser Pseudo-Befehl signalisiert das Ende eines Makros.

`[label] makroname [par[,par[,...]]]`

Makroaufruf: Dieser Pseudo-Befehl wird für den Aufruf (oder Erweiterung) eines Makros benutzt. Das Label ist optional. Wenn angegeben, wird die programmrelative Adresse des ersten Bytes der Makroerweiterung unterstellt. *Makroname* ist der dem Makro gegebene Name. Aktuelle Parameter werden in einer durch Komma getrennten Liste im Operandenfeld angegeben. Parameter sind bloße Zeichenfolgen, die entsprechende Platzhalter im Makro selbst ersetzen. Wenn Leerzeichen, Komma oder Semikolons im Parameter sind, müssen sie mit Anführungszeichen (") oder Hochkomma (') gekennzeichnet werden. Wenn diese Zeichen innerhalb von Zeichenketten vorkommen, müssen sie doppelt angegeben werden. Fehlende Parameter werden als leere Zeichenkette interpretiert. Zwei aufeinanderfolgende Kommas kennzeichnen einen fehlenden Parameter. Ein Parameter fehlt auch, wenn die Parameterliste nicht groß genug ist.

Ausdrücke

Ausdrücke können im Operandenfeld einiger Maschineninstruktionen oder Pseudo-Befehle erscheinen. Für die richtige Stellung der Ausdrücke in den Maschineninstruktionen siehe auch die Instruktionstabellen in Anhang D. Die Ausdrucksauswertung erzeugt immer einen binären 16-Bit-Wert. Wenn der Ausdruck im Operandenfeld eines Feldes erscheint, wird ihr Wert in die Objektdatei gebracht. Wenn die Instruktion weniger als 16 Bit erfordert, werden die höherwertigen Bits abgeschnitten. Ähnlich produzieren auch die Ausdrücke in den Pseudobefehlen DW und DB Werte in der Objektdatei.

Regeln für die Verschiebung

Der Wert eines Ausdrucks ist entweder absolut oder programmrelativ, abhängig davon, ob und wie die Symbole benutzt werden.

Programmrelative Teile in einer Objektdatei werden durch den Linker in absolute umgewandelt, sobald er einmal die absolute Adresse, an der das Modul stehen soll, errechnet hat, wohingegen absolute Teile unverändert geladen werden.

Ein Label hat immer einen programmrelativen Wert, das heißt, daß sie die programmrelative Adresse des nächsten zu assemblierenden Teils im Pro-

gramm unterstellt. Eine numerische Konstante ist aber absolut. Das Attribut für die Verschiebung eines Ausdrucks hängt von den Attributen seiner primären Terme und ihrer Kombination ab. Die folgende Tabelle zeigt die Regeln zur Bestimmung der Attribute zur Verschiebung eines Ausdrucks:

KOMBINATION	ERGEBNIS
abs ? abs	abs
abs + rel	rel
abs ? rel	Fehler
rel + abs	rel
rel - abs	rel
rel - rel	abs
rel == rel	abs
rel < rel	abs
rel <= rel	abs
rel != rel	abs
rel > rel	abs
rel >= rel	abs
rel ? rel	Fehler

Ein Fragezeichen in der Liste steht für jeden anderen Operator, als denjenigen, der ausdrücklich für jede links- und rechtsseitige Kombination gezeigt wird.

Normalerweise nehmen nur 16-Bit-Objektfelder verschiebare Ausdrücke auf. Ein verschiebarer Ausdruck kann jedoch für ein PC-relatives Feld erscheinen; das heißt, ein Feld, das einen Offset mit Vorzeichen hat, der von der CPU zum Programmzähler addiert wird, um die effektive Adresse zu erhalten. Der Z80-Befehl JR (jump relative, relativer Sprung) ist ein Beispiel für die PC-relative Adressierung. In diesen Fällen nimmt MAC die Ausdrücke als Zieladresse. Vom Ausdruck subtrahiert er den Positionszähler und die Instruktionslänge, wobei er ihn zu einem absoluten Wert des folgenden Ausdrucks umwandelt. Wenn aber der Ausdruck eine absolute Adresse ergibt, nimmt MAC an, daß der Programmierer einen Offset zur gegenwärtigen Position anzeigen wollte; daher subtrahiert er nur die Instruktionslänge und stellt damit in Rechnung, daß die CPU den Offset nach Weiterstellung des PC anwendet.

Zahlen

Zahlen müssen Integer-Werte sein. Sie werden als dezimal angesehen solange nicht unmittelbar danach ein O oder Q (oktal) oder H (hexdazimal) folgt.

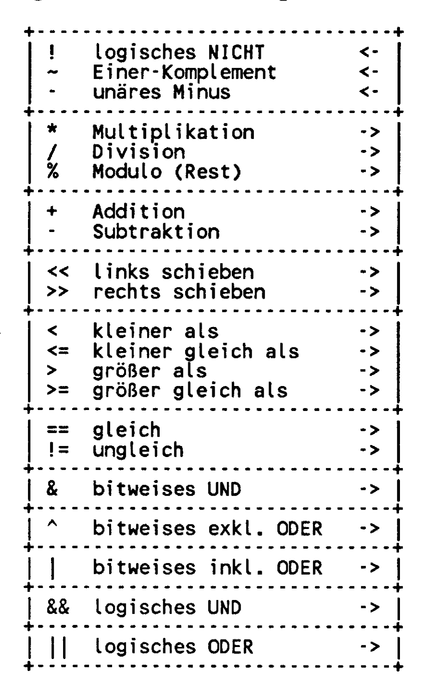
Das erste Zeichen einer Zahl muß eine Ziffer sein. Eine führende Null kann erforderlich sein, damit hexadezimale Zahlen dieser Regel genügen. Zahlen werden in 16-Bit-Werte umgewandelt und dann mit dem Rest des Ausdrucks (sofern vorhanden) kombiniert.

Symbole

Symbole in einem Ausdruck müssen entweder irgendwo definiert oder als extern deklariert worden sein. Ein Symbol, das in einem Operandenfeld erscheint, kann durch ein Nummernzeichenpaar (##) abgeschlossen werden. Die Anweisung EXT erklärt Symbole als extern. Externe Symbole besitzen das programmrelative Attribut. Symbole, die mit den Pseudobefehlen SET und EQU definiert worden sind, wird das Attribut für die Verschiebung zugewiesen, das die Ausdrücke ihrer Operandenfelder besitzen.

Operatoren

Die Ausdrucksoperatoren von Small-Mac sind ein Subset von denen der Sprache C und folgen den gleichen Vorrang- und Ordnungsregeln. Sie sind in der folgenden Tabelle aufgeführt.



!	logisches NICHT	<-
~	Einer-Komplement	<-
-	unäres Minus	<-
*	Multiplikation	->
/	Division	->
%	Modulo (Rest)	->
+	Addition	->
-	Subtraktion	->
<<	links schieben	->
>>	rechts schieben	->
<	kleiner als	->
<=	kleiner gleich als	->
>	größer als	->
>=	größer gleich als	->
==	gleich	->
!=	ungleich	->
&	bitweises UND	->
^	bitweises exkl. ODER	->
|	bitweises inkl. ODER	->
&&	logisches UND	->
||	logisches ODER	->

Operatoren mit der höchsten Priorität stehen an der Spitze und alle Operatoren im gleichen Kasten haben auch die gleiche Priorität. Pfeile zeigen die Reihenfolge der Anordnung. Man kann Klammern benutzen, um die Anordnung zu kontrollieren. Jede Schachtelungstiefe ist erlaubt.

Die aktuelle Instruktionsadresse

Man kann das Dollarzeichen als Label für die Adresse der aktuellen Instruktion auffassen. Es hat das Attribut programmrelativ.

Die Makro-Möglichkeiten

Quellzeilen, die sich zwischen den Makro-Pseudobefehlen MACRO und ENDM befinden, sind das eigentliche Makro. Während des ersten Assemblerlaufs werden sie in einem Puffer untergebracht. Beim zweiten Lauf werden an dem Punkt, an dem der Makroname im Operationsfeld gefunden wird, die Makroquellzeilen eingefügt. Dies wird als Makroerweiterung oder als Makroaufruf bezeichnet. Der erste Term leitet sich davon ab, daß eine einzige Instuktion zu einem ganzen Instruktionssatz erweitert wird. Der zweite Term macht sich die Analogie zu den Unterprogrammaufrufen zu eigen. Makros werden in der Tat auch als offene oder Inline-Unterprogramme bezeichnet. Makroaufrufe müssen ihrer Definition in der Quelldatei folgen.

Eine Schachtelung von Makrodefinitionen und Makroaufrufen ist nicht erlaubt. Wenn mehr als eine Makrodefiniton den gleichen Namen hat, wird nur die erste benutzt.

Ersetzung der Parameter

Parameter können bei jedem Makroaufruf angegeben werden, damit der erweiterte Code sich für den speziellen Aufruf maßschneidern läßt. Man kann für den ersten Parameter ganz einfach ?1 in das Makro einfügen, ?2 für den zweiten und ?0 für den zehnten und letzten. Es sind höchstens zehn Parameter erlaubt. Parameter in einem Makroaufruf werden durch ihre Position identifiziert und durch Komma getrennt. Aufeinanderfolgende Kommas oder fehlende Parameter am Ende erzeugen keine Ersetzung. Das heißt, der für die Substitution vorgesehene Parameter wird aus dem erweiterten Text entfernt. Wenn man ein ? im erweiterten Text braucht, muß man ?? codieren. Anführungszeichen oder Hochkommas können einen Parameter einschließen, der Leerzeichen enthält. Müssen diese Zeichen auch innerhalb des Makros erscheinen, müssen sie doppelt im Makro codiert werden.

Die Ersetzung der Parameter erfolgt ohne Rücksicht auf den Kontext. Daher erfolgen die Ersetzungen auch innerhalb von Zeichenketten mit Anführungszeichen, Kommentaren und sogar Symbolen und Mnemonics. Man kann dieses einfache Konzept zu seinem Vorteil verwenden.

Lokale Label

Zehn Label, die lokal zu jeder Makroerweiterung sind, können im Makro als @0 bis @9 gekennzeichnet werden. Das erste dieser vom Assembler gefundenen Label erscheint im erweiterten Text als @1, das zweite als @2, und so weiter. Diese Folge erhöht sich während des Programms fortlaufend, so daß gewährleistet ist, daß jedes Label und seine Verweise nur einmal vorkommen. Dies vermeidet den Fehler "redundant definition", wenn dasselbe Makro wiederholt aufgerufen wird.

Laden und Ausführen

Manchmal müssen Programme entwickelt werden, die an anderen und nicht den üblichen CP/M-TPA-Adressen starten. Häufig sind solche Programme besondere Gerätetreiber oder andere BIOS-Erweiterungen, die gewöhnlich bei einem Kaltstart aufgerufen werden. Small-Mac hat deswegen ein besonderes Laden-und-Ausführen-Programm, den Lader LGO. LGO lädt und übergibt wahlweise die Kontrolle an Programme, die von LINK mit dem besonderen Laden-und-Ausführen-Format erzeugt wurden.

Der Schalter -G# sorgt dafür, daß LINK ein Laden-und-Ausführen-Programm erzeugt. Das Nummernzeichen (#) gibt die hexadezimale Adresse an, an der das Programm starten soll. Laden-und-ausführen-Programme enthalten eine RET-Anweisung zum CP/M, so daß Versuche, sie als normale CP/M-Befehle zu benutzen (durch Umbenennung der Dateinamenerweiterung) zum Scheitern verurteilt sind. Nach der RET-Anweisung folgen Definitionen für die Startadresse, die Länge in Bytes und die Startadresse. Diese Information wird von LGO benutzt, um genau die richtige Anzahl der Bytes an die richtige Adresse zu laden und um (wahlweise) mit der Ausführung am richtigen Ort beginnen zu können.

Es liegt in der Verantwortung des Progammierers sicherzustellen, daß dies während folgender Betriebssystemsoperationen keine Probleme verursacht. Man sollte in den CP/M-Handbüchern die richtige Technik nachlesen.

Die Benutzerschnittstelle

Der Aufruf von Small-Mac-Programmen besteht aus:

1. dem Programmnamen mit optionaler Laufwerksangabe im Standard-CP/M-Format,
2. Umlenkungsangaben für Standard-Ausgabedateien und
3. Schaltern, die den Ablauf des Programms kontrollieren.

Standardein- und -ausgabe

Small-Mac-Programme unterstüzten die Unix-ähnliche Umlenkung von Standarddateien. Eine Standarddatei ist eine Datei, die automatisch bei der Programmausführung geöffnet wird. Üblicherweise wird die Standardeingabe der Tastatur zugewiesen und die Standardausgabe dem Bildschirm.

Man kann die Standardeingabe von der Tastatur durch ein < in der Befehlszeile mit nachfolgender neuer Quelle auf diese Quelle umlenken. Dies kann eine Datei (komplett mit Erweiterung und Laufwerksangabe) oder ein logisches Gerät wie RDR: sein.

Genauso kann man auch die Standardausgabe vom Bildschirm durch ein > in der Befehlszeile umlenken. Wenn die Standardausgabe durch ein >> umgelenkt wird (zum Beispiel >>DATEI3), wird die Ausgabe an die schon vorhandenen Daten angehängt, was auch immer schon vorhanden gewesen sein mag. Wenn die Datei noch nicht besteht, wird sie angelegt und >> unterscheidet sich nicht von >.

Beide Umlenkungsanweisungen können gleichzeitig in jeder Position nach dem Programmnamen in der Befehlszeile erscheinen. Man sollte darauf achten, nicht Eingabe und Ausgabe der gleichen Datei zuzuweisen; das Ergebnis wäre eine zerstörte Datei.

Small-Mac-Programme benutzen die Standardeingabe nur für Antworten auf Fehlermeldungen über die Tastatur, so daß für eine Umlenkung keine Notwendigkeit bestehen sollte. Für Fehlermeldungen und Listings wird die Standardausgabe benutzt, die somit standardmäßig zum Bildschirm geht.

Parameter in der Befehlszeile

Schalter bestehen aus einem Bindestrich, gefolgt von einem oder zwei Buchstaben und eventuell von einem numerischen Wert. Die Buchstaben für die Schalter sind nach mnemonischen Aspekten gewählt worden. Außer bei LIB können alle Schalter in Small-Mac-Programmen in jeder Position hinter dem Dateinamen erscheinen. LIB benutzt nur einen Schalter, der als erster in der Befehlszeile erscheinen muß.

Small-Mac-Programme interpretieren Buchstaben, die keine Schalter sind, als Dateinamen. Die Reihenfolge kann einen Unterschied bewirken. Die Details sind in den einzelnen Programmbeschreibungen weiter unten aufgeführt

Bedienungshinweise

Damit man sich an die verschiedenen Schalter und Parameter bei der Benutzung von Small-Mac-Programmen besser erinnern kann, erscheint auf dem Bildschirm immer dann ein Bedienungshinweis, wenn Small-Mac ohne Schalterbuchstaben oder mit einem nicht definierten Schalter aufgerufen wird. Wenn man Hilfe braucht, um sich an die verschiedenen Schalter zu erinnern, braucht man nur den Programmnamen gefolgt von einem Bindestrich einzugeben. Bedienungshinweise zeigen die Aufruf-Syntax der Small-Mac-Programme; sie haben die Form

```
usage: <program> <switch>... <file>...
```

wobei <program> der Programmname ist, <switch> ist ein Schalter und <file> ist ein Dateiname. Die Zeichen < und > sind nicht Teil der Syntax. Sie zeigen nur an, daß ein gültiger Term eingeschlossen wurde.

Angaben zur Umlenkung werden im Bedienungshinweis nicht gezeigt, da sie bei allen Programmen gleich sind; ihre Verfügbarkeit kann angenommen werden. Eckige Klammern im Bedienungshinweis geben optionale Felder an. Die Klammern sind nicht Teil der Kommandos.

Programme nehmen einen Standardablauf, wenn ein optionaler Schalter fehlt. Der Ablauf orientiert sich an der üblichen Vorgehensweise, so daß Schalter nur in Sonderfällen gebraucht werden. LIB ist eine Ausnahme, da ihm immer gesagt werden muß, wie es zu verfahren hat.

Punkte erscheinen in der Benutzungsmeldung um anzuzeigen, daß ein gegebener Feldtyp mehr als einmal im Befehl erscheinen kann. Die Punkte selbst sind nicht Teil des Kommandos.

Worte oder Kombinationen aus Wörtern werden als gültige Namen für besondere Parameterarten verwendet. Zum Beispiel steht *source* für eine Assembler-Eingabedatei.

Fehlerbehandlung

Alle Small-Mac-Programme schicken Fehlermeldungen zur Standardausgabe. Bei einem fatalen Fehler bricht das Programm mit hörbarem Alarm ab. Wenn möglich, versucht das Programm ganz durchzulaufen, ehe es ab-

bricht. Einige Fehler jedoch (zum Beispiel Fehler in der Befehlszeile) können zum sofortigen Ende des Programms führen.

Zwei Fehler werden vom Small-C-Laufzeitsystem und nicht vom Programm selbst gefunden:

1. R, Fehler bei der Umlenkung, der anzeigt, daß versucht wurde, die Standardeingabe zu einer nicht existierende Datei umzulenken. Dies sollte eigentlich nicht vorkommen, da es keinen Grund gibt die Standardeingabe von Small-Mac umzulenken.
2. M, Speicherzuordnungsfehler, der anzeigt, daß versucht wurde, mehr Speicher zuzuordnen als verfügbar ist.

Programmkontrolle

Man kann jedes Programm im Small-Mac-Paket während des Laufs abbrechen. Um ein Programm zeitweise anzuhalten, kann man Control-S eingeben. Der Druck auf eine beliebige andere Taste nimmt die Programmausführung wieder auf. Zum Programmabbruch kann man Control-C eingeben.

MAC: Der Small-Mac-Makroassembler

Bedienung

```
MAC [-L] [-NM] [-P] [-S#] [object] source...
```

`-L`	Assemblerlisting erzeugen
`-NM`	keine Makroverarbeitung
`-P`	Pause bei Fehlern
`-S#`	die Größe der Symboltabelle auf # Symbole einstellen
`object`	Name der Objektdatei
`source...`	Namen der Quelldateien

Quelldateien

Man muß mindestens eine Quelldatei in der Befehlszeile angeben. Wenn mehr als eine angegeben wird, werden sie in der angegebenen Reihenfolge zu einem Modul assembliert. Man kann bei den Quelldateien auch eine Laufwerksbezeichnung angeben, um Quelldateien von verschiedenen Laufwerken zu lesen. Wenn kein Laufwerk angegeben wird, wird das Standardlaufwerk genommen. Wenn die Quelldatei nicht gefunden wird, bricht MAC mit einer Fehlermeldung ab. Die Standard- und einzige erlaubte Dateinamenserweiterung ist MAC.

Die Objektdatei

Man kann eine Objektdatei angeben. Wenn keine angegeben wird, wird der Objektcode auf dem Standardlaufwerk in einer Datei abgelegt, die den gleichen Namen trägt, wie die erste Quelldatei, aber mit der Erweiterung REL. Man kann für die Objektdatei ein Laufwerk angeben, um die REL-Datei auf diesem Laufwerk zu speichern. Wenn kein Laufwerk angegeben ist, wird das Standardlaufwerk genommen. Die Objektdatei muß die Erweiterung REL haben, zur Unterscheidung von Quelldateien. Der Modulname in der Objektdatei wird aus den ersten sechs Zeichen des Objektdateinamens gebildet.

Das Assembler-Listing

Ein Assembler-Listing wird nur erzeugt, wenn der Schalter -L in der Befehlszeile angegeben wurde. Das Listing und die Fehlermeldungen werden zur Standardausgabe geschickt und gehen daher zum Bildschirm, sofern nicht umgelenkt wird.

Fehlermeldungen erscheinen in der gleichen Zeile wie der Fehler. Wenn jedoch kein Listing erzeugt werden soll, wird jede fehlerhafte Zeile vor der eigentlichen Fehlermeldung gedruckt.

Die Listingausgabe ist für Seiten mit einer Höhe von 11 Zoll vorgesehen. Man sollte entweder einen breiten Drucker benutzen oder die Listings in komprimierter Schrift ausdrucken.

Jede Zeile im Listing enthält (von links nach rechts):

- die dezimale Quellzeilennummer
- den aktuellen hexadezimalen Positionszählerwert
- den durch die aktuelle Quellzeile generierten Objektcode
- die Quellzeile

Die im Programm verschiebaren Teile sind in der Objektspalte mit einem Hochkomma gekennzeichnet. Alles andere ist absolut. Wenn der Objektcode nicht in den zugewiesenen Platz paßt, werden Überlaufzeilen gedruckt.

Eine sortierte Symboltabelle wird am Ende des Listings gedruckt. Jede Zeile zeigt den Symbolwert, das Attribut für die Verschiebung, das Symbol und den Symboltyp. Symboltypen werden durch folgende nachgestellte Zeichen markiert:

:	Label
::	Einsprungspunkt
##	externe Referenz

Übergehen von Makros

Man kann den Schalter -NM mit der Bedeutung "keine Makros" ("no macros") angeben, wenn die Makroverarbeitung nicht erwünscht ist. Dies beschleunigt den Assembler um 13%. Makroverarbeitung wird für Programme, die vom Small-C-Compiler erzeugt werden, nicht gebraucht.

Pause bei Fehlern

Beim Schalter -P hält MAC an, nachdem er die Fehler für jede Zeile ausgegeben hat. Er wartet dann solange, bis RETURN eingegeben wird.

Größe der Symboltabelle

Der Schalter -S# bestimmt die Größe der Symboltabelle. Das Nummernzeichen steht für eine dezimale Integer ohne Vorzeichen. Sie gibt die maximale Anzahl der Symbole in der Tabelle an. Da die Geschwindigkeit mit

der Annäherung an die Kapazitätsgrenze abnimmt, sollte man etwas Platz freilassen. Die Standard-Tabellengröße beträgt 500 Symbole.

Der Speicherplatz, der nach der Zuordnung der Symboltabelle noch frei ist, wird als Makropuffer verwendet. Je größer die Symboltabelle ist, desto weniger Platz bleibt für die Makrodefinitionen; je kleiner die Symboltabelle ist, desto mehr Platz bleibt für die Symboltabelle.

Wenn man ein Assemblerprogramm mit ungefähr 400 Symbolen oder mehr benötigt, sollte man -S# benutzen um die Größe der Symboltabelle zu erhöhen. Wenn man auf der anderen Seite die Fehlermeldung "Macro Buffer Overflow" erhält, sollte man die Symboltabelle verkleinern.

Beispiele

BEFEHL BESCHREIBUNG

`MAC PROG`

Assembliert PROG.MAC, erzeugt PROG.REL auf dem Standardlaufwerk, produziert kein Listing und hält bei Fehlern nicht an.

`MAC PROG PROG2 -L -P`

Assembliert PROG.MAC und PROG2.MAC, erzeugt PROG. REL auf dem Standardlaufwerk. Gibt ein Listing auf dem Bildschirm aus und hält bei Fehlern an.

`MAC P1 P2 P3 B:P.REL -NM`

Assembliert P1.MAC, P2.MAC und P3.MAC und erzeugt P.REL auf Laufwerk B. Produziert kein Listing, hält bei Fehlern nicht an und verarbeitet keine Makros.

`MAC PROG -L >LST:`

Assembliert PROG.MAC und PROG2.MAC und erzeugt PROG.REL auf dem Standardlaufwerk. Gibt ein Listing auf LST: aus und hält bei Fehlern nicht an.

Normale Meldungen

MELDUNG ERKLÄRUNG

`Waiting...`

MAC wartet auf ein RETURN von der Tastatur.

Fehlermeldungen

MELDUNG ERKLÄRUNG

`Backward Movement`
: Ein ORG würde den Positionszähler des Assemblers rückwärts bewegen.

`Bad Data`
: Ein DW oder DB gibt ungültige Daten an.

`Bad Expression`
: Im Operandenfeld steht ein ungültiger Ausdruck.

`Bad Label`
: Im Feld Symbol/Label steht ein ungültiges Label.

`Bad Operation`
: Ein Mnemonik kann in der Maschineninstruktionstabelle nicht gefunden werden.

`Bad Parameter`
: Ein Makroaufruf gibt zu viele Parameter an.

`Bad Symbol`
: Ein ungültiges Symbol ist gefunden worden.

`xxxxxxxx - Can't Open`
: Die Datei xxxxxxx kann nicht geöffnet werden.

`Close Error`
: Eine Datei kann nicht richtig geschlossen werden.

`Invalid Extension`
: Eine Datei aus der Befehlszeile enthält eine falsche Erweiterung.

`Macro Buffer Overflow`
: Der Makrotext-Puffer ist übergelaufen.

`Missing END`
: Bei einer Eingabedatei fehlt der Pseudobefehl END.

`Missing ENDM`
: Innerhalb einer Makrodefinition wurde das Ende einer Quelldatei entdeckt.

`Redundant Definition`
: Das gleiche Symbol erscheint mehr als einmal im Feld Symbol/Label. Nur SET ist für eine Neudefinition von Symbolen erlaubt und nur bei denen, die ursprünglich von ihm definiert wurden.

`Relocation Error`
: Ein 8-Bit-Feld wurde zu einem programmrelativen Wert berechnet oder der Ausdruck bei einer END-Anweisung ist nicht programmrelativ.

`Symbol Table Overflow`
Es passen keine Symbole mehr in die Symboltabelle.

`xxxxxxxx - Too Long`
Die Befehlszeile enthält einen Dateinamen, der zu lang ist.

`Undefined Symbol`
Das Operandenfeld enthält einen Verweis auf ein undefiniertes Symbol.

`Write Error in REL File`
Bei der Ausgabe der Objektdatei ist ein Fehler aufgetreten. Wahrscheinlich ist auf dem Laufwerk kein Platz mehr.

LNK: Der Small-Mac Linker

Bedienung

`LNK [-B] [-G] [-M] program [module/library...]`

-B Es soll ein großes Programm (Big) gelinkt werden. Der ganze verfügbare Speicherplatz wird für die Symboltabelle reserviert und das Programm wird komplett auf dem Laufwerk gepuffert.

-G# Eine LGO-Datei (Laden-und-ausführen) ausgeben, für Ausführung an Adresse #.

-M Die Aktivitäten des Linkers beobachten (Monitor).

program Ein zu linkendes Programm.

module/library...
  Eine Liste von keiner oder mehreren Dateien und/oder Bibliotheksdateien (.LIB).

Funktionale Beschreibung

LNK führt primär drei Aufagben durch: es kombiniert separat assemblierte Module in ein einziges Programm, löst externe Referenzen auf und addiert die Basisadresse von jedem Modul zu allen darin enthaltenen programmrelativen Teilen und wandelt sie dadurch in absolute Adressen um.

Diese Arbeit wird in zwei Durchgängen ausgeführt. Zuerst wird die Befehlszeile nach Modul- und Bibliotheksnamen durchsucht. Jedes in der Befehlszeile aufgeführte Modul wird in einen Puffer hinter das vorige Modul geladen. Während jedes Modul geladen wird, wird eine temporäre Datei mit Zeigern auf programmrelative Teile für die Benutzung im zweiten Durchgang in eine Referenzdatei (.R$) geschrieben. Ebenso werden Einsprungspunkte und externe Referenzen in eine Symboltabelle geladen, um die externen Referenzen aufzulösen.

Die Symboltabelle ist als zwei miteinander verkettete Listen aufgebaut, angeordnet in alphanumerischer Reihenfolge, eine für externe Referenzen und eine für Einsprungspunkte. Jede Eintragung in der Symboltabelle enthält einen Kettenzeiger, den Namen eines Symbols und eine 16-Bit-Adresse. Im Falle eines Einsprungpunktes ist der Wert die Einsprungsadresse. Im Falle einer externen Referenz ist es die Adresse auf den Kettenkopf von Adressen für dieses Symbol. LNK löst externe Referenzen auf, indem jeder Verweis in der Kette durch den entsprechenden Wert des Einsprungpunktes ersetzt wird.

Nachdem jedes Modul geladen wurde, versucht LNK alle externen Referenzen, die mit Einsprungspunkten im neuen Modul übereinstimmen, aufzulösen. Sobald jede externe Referenz aufgelöst wird, wird ihr Platz in der Symboltabelle für spätere Benutzung freigegeben. Neue externe Referenzen benutzen diesen freien Platz, bevor die Tabelle erweitert wird. Einsprungspunkte müssen in der Tabelle verbleiben, für den Fall, daß neue Module auf sie Bezug nehmen.

Wenn LNK eine Datei mit der Erweiterung LIB findet, wird vermutet, daß es sich um eine Bibliothek handelt. In diesem Fall erfolgt eine Suche nach den Bibliotheksmodulen, die Einsprungspunkte enthalten, die bis jetzt noch nicht aufgelöst werden konnten. Diese Suche wird durch die Tatsache erleichtert, daß jeder in dem Modul enthaltenene Einsprungspunkt am Modulanfang aufgeführt ist. Wenn keine Übereinstimmung gefunden wird, wird das Modul ausgelassen. Zur Beschleunigung dieses Ausslassungsprozeß wird eine Indexdatei benutzt (NDX), um die Position des nächsten Moduls zu erhalten. LNK geht sofort zum nächsten Modul, ohne das unerwünschte durchzulesen. Wenn die Suche beendet ist, wird die Bibliothek noch einmal rückwärts durchsucht, um Rückverweise auflösen zu können. Dies wird solange wiederholt, bis keine Module mehr zu laden sind. LNK fährt dann fort, die Befehlszeile nach weiteren Modulen und/oder Bibliotheken zu durchsuchen.

Es reicht aus, ein Symbol als extern zu erklären, damit das Modul geladen wird. Es muß nicht ausdrücklich darauf Bezug genommen werden.

Wenn die Befehlszeile ausgewertet ist, beginnt LNK mit der zweiten Phase. Die Referenzdatei wird geschlossen und wieder für die Eingabe geöffnet. Dann wird der gepufferte Objektcode zur Ausgabedatei geschrieben (entweder COM oder LGO). Dabei wird jede Adresse mit der Adresse der Referenzdatei verglichen. Bei jeder Übereinstimmung ist ein relativer Teil gefunden worden, also addiert LNK einen Offset und macht so einen absoluten Wert daraus. Die nächste Referenzadresse wird gelesen und der Prozeß wiederholt sich. Um die Diskettenkopfbewegungen während des zweiten Durchgangs gering zu halten, werden für die Referenzdatei zusätzliche Puffer benutzt.

Der Schalter -B

Normalerweise benutzt LNK allen verfügbaren Speicher zur Pufferung des Programmtextes (Objekt) und für die Symboltabelle. Es kann jedoch sein, daß nicht genug Platz für LNK, seine Symboltabelle und für das geladenen Programm vorhanden ist. Wenn LNK bemerkt, daß das zu ladende Programm nicht in den Speicher paßt, wird zusätzlich eine temporäre Datei

mit Erweiterung ".0$" angelegt. Die Verarbeitung geht dann wie vorher weiter, nur daß die jetzt folgenden Module in diese Datei geladen werden. Dies verlangsamt den Ladeprozeß beträchtlich, geschieht aber auch nur bei größeren Programmen.

Das untere Ende des Speichers wird für den Programmtext und das obere Ende für die Symboltabelle verwendet. Bei der Verarbeitung wächst jedes in die Richtung des anderen. Einen Überlauf erhält man dann, wenn sich das nächste zu ladende Modul mit einer imaginären Reserve von 200 Einträgen am Ende der Symboltabelle überlappen würde. Sogar nach Auslagerung auf Diskette kann die Symboltabelle überlaufen, wenn die Reserve erschöpft ist. An diesem Punkt bricht LNK mit der Meldung "Must Specify -B Switch" ("Schalter -B muß verwendet werden) ab. Wenn man LNK mit dem Schalter -B aufruft, wird sofort auf Diskette ausgelagert und der verfügbare Speicher wird nur für die Symboltabelle benutzt. Dies sollte das Laden eines Programms von jeder Größe erlauben.

Der Schalter Laden-und-Ausführen

Der Schalter -G# veranlaßt LNK, anstelle einer COM-Datei eine LGO-Datei zu erzeugen. Die Adresse, an der das Programm startet, wird durch eine hexadezimale Zahl mit vorangestelltem # gekennzeichnet. Der LGO-Lader muß benutzt werden, um das Programm zu laden und/oder auszuführen.

Der Monitor-Schalter

Durch den Schalter -M können die Lade- und Link-Aktivitäten beobachtet werden. Auf der Standardausgabe wird eine Liste der geladenen Module ausgegeben, ihre Größe und ihre Ladeadresse (Programmrelativ und absolut).

Programm- und Modulnamen

Wenigstens ein Modulname (REL-Datei) muß angegeben werden. Ein Dateiname ohne Erweiterungen wird als Modulname interpretiert. Das erste Modul bestimmt den Namen des ausgegebenen Programms. Nachfolgende Module werden zum Programm-Modul geladen, haben jedoch keinen Einfluß auf den Programmnamen. Wenn ein Modulname eine Erweiterung hat, muß sie REL lauten. Wenn ein Laufwerk angegeben ist (zum Beispiel B:), wird es für die Suche nach der Eingabedatei benutzt. Die Ausgabe erfolgt jedoch immer auf das Standardlaufwerk.

Bibliotheksnamen

Bibliotheken werden durch die Erweiterung LIB gekennzeichnet, die zusammen mit dem Dateinamen angegeben werden muß. Mit einer Laufwerksangabe kann angegeben werden, wo nach der Bibliothek und dem Index gesucht werden soll. LNK erfordert, daß eine Indexdatei mit gleichem Namen sich auf dem gleichen Laufwerk befindet, aber mit der Erweiterung NDX

Die Bibliothek C.LIB enthält die Standard-Laufzeitfunktionen von Small-C. Jedesmal, wenn man ein Small-C Programm linkt, muß man C.LIB angeben.

Das besondere Modul END

Um das Linken von Small-C-Programmen zu erleichtern, achtet LNK auf ein Modul mit Namen END. Dieses Modul in C.LIB muß immer zuletzt geladen werden. Dies deshalb, weil es Anweisungen enthält, die den Beginn des freien Speichers kennzeichnen, bevor das Programm ausgeführt wird. Wenn LNK ein Modul mit diesem Namen lädt, wird es übergangen und am Ende des ersten Durchganges geht LNK zurück und lädt es als letztes. Wenn LNK mehr als ein Modul mit Namen END findet, wird nur das letzte geladen. Dies kann nützlich sein, wenn man sich seine eigene Bibliothek zusammenstellen möchte.

Beispiele

BEFEHL KOMMENTAR

`LNK PROG -GE000`

Lädt PROG.REL, erzeugt PROG.LGO zur Ausführung an Adresse E000.

`LNK P1 P2 C.LIB`

Linkt P1.REL, P2.REL und alle erforderlichen Module aus C.LIB und erzeugt P1.COM zur Ausführung als Standard-CP/M-Befehl.

`LNK -B -M >LST: ABC C.LIB`

Linkt ABC.REL und alle erforderlichen Module aus C.LIB und erzeugt ABC.COM. ABC ist sehr groß, daher wird es gänzlich auf die Diskette geladen. Aller verfügbarer Speicherplatz bleibt für die Symboltabelle übrig. Der Linkprozeß wird auf dem LST-Gerät angezeigt.

Normale Meldungen

MELDUNG ERKLÄRUNG

`xxxx Bytes at yyyy zzzz mmmmmm`
: Modul mmmmmm mit Länge xxxx wird an relative Adresse yyyy (absolut zzzz) geladen. Erscheint nur, wenn -M angegeben wurde.

`xxxx Byte Buffer`
: Der verfügbare Speicherplatz für die Symboltabelle und den Objektpuffer beträgt xxxx. Erscheint nur, wenn -M angegeben wurde.

`xxxx Bytes (hex)`
: Die Programmgröße ist hexadezimal xxxx.

`xxxxx Bytes (dec)`
: Die Programmgröße ist dezimal xxxxx.

`xxxx Overflow Point`
: Die relative Adresse des ersten Moduls beim Überlauf ist xxxx. Erscheint nur, wenn LNK mit dem Symbol DEBUG kompiliert und -M angegeben wurde.

`Resolving xxxxxx to yyyy`
: Um die externe Referenz xxxxxx aufzulösen startet LNK mit dem Kopf der Kette bei yyyy. Erscheint nur, wenn LNK mit dem Symbol DEBUG kompiliert und -M angegeben wurde.

`Start in xxxxxx`
: Eine Startadresse für Modul xxxxxx wurde angegeben.

Fehlermeldungen

MELDUNG ERKLÄRUNG

`Abnormal End of REL File`
: Das Ende eines Moduls oder einer Bibliothek wurde erreicht, ohne das das richtige Dateiendezeichen gefunden wurde.

`xxxxxxxx - Can't Open`
: Die angegebene Datei kann nicht geöffnet werden.

`Close Error`
: Eine Datei kann nicht geschlossen werden.

`Corrupt Library or Index`
: Bei dem Versuch, das nächste Modul in der Bibliothek zu finden, trat ein Suchfehler auf.

`Corrupt Module`
: In einem Modul oder einem Bibliotheksteil wurde ein nicht erkennbares Linkmodul gefunden.

`Error Reading xxxxxxxx`
Beim Lesen der genannten Datei trat ein Ein-/Ausgabe-Fehler auf.

`Error Writing xxxxxxxx`
Beim Schreiben auf die genannte Datei trat ein Fehler auf. Höchstwahrscheinlich bedeutet dies, daß kein Platz mehr auf der Diskette ist.

`Invalid Extension`
Eine Datei aus der Befehlszeile enthält eine ungültige Erweiterung.

`Must Specify -B Switch`
Es gibt nicht genug freien Speicherplatz, um die Symboltabelle und das Programm zu laden. LNK erneut mit dem Schalter -B aufrufen.

`Premature End of Index`
Der Index einer Bibliothek enthält nicht genug Einträge.

`Redundant: xxxxxx`
Das genannte Symbol wird mehr als einmal als Einsprungspunkt genannt.

`Seek Error in xxxxxxxx`
Bei dem Versuch, ein programmrelatives Modul in einer Überlaufdatei zu finden, trat ein Suchfehler auf. Ursache könnte ein Problem mit der Überlaufdatei oder ein logischer Fehler in LNK sein.

`xxxxxxxx - Too Long`
Die Befehlszeile enthält einen zu langen Dateinamen.

`Unresolved: xxxxxx`
Es wurde kein Einsprungspunkt gefunden, der mit der externen Referenz übereinstimmte.

`Unsupported Link Item`
Ein Eingabemodul enthält zwar ein erkennbares Modul, es ist aber nicht im Microsoft-Link-Format.

LGO: Der Small-Mac-Lader

Bedienung

`LGO [-G] [-M] program`

-G	Programm nach dem Laden ausführen.
-M	Ladeadresse, Größe und Anfangsadresse anzeigen.
program	Dateiname des zu ladenden Programms.

Beschreibung

LGO ist ein sehr einfacher Lader, um LGO-Dateien an ihre geplante Adresse zu laden und sie wahlweise zu starten. Sein primärer Zweck ist es, die bequeme Installation von Betriebssystemserweiterungen beim Kaltstart zu erlauben.

Wenn der Schalter -G# mit LNK benutzt wird, wird eine besondere LGO-Datei anstelle der normalen COM-Datei ausgegeben. Dateien zum Laden und Ausführen haben folgendes Format:

TEIL	BYTE
RET-Anweisung	1
Startadresse	2
Basisadresse	2
Programmgröße	2
Objektprogramm	<Größe>

Die RET-Anweisung dient zwei Zwecken. Sie indentifiziert Dateien, die das Laden-und-Ausführen-Format haben und erzeugt einen Ausgang für die Fälle, in denen jemand versucht, die Datei in eine mit COM-Erweiterung umzubenennen, um sie dann als CP/M-Befehl auszuführen.

Die Basisadresse und Größe sagen LGO genau, wohin das Programm zu laden ist und wieviele Bytes geladen werden. Nur die angezeigte Zahl der Bytes wird geladen. Dies Zahl wird von LNK errechnet, wenn die LGO-Datei erzeugt wird. Man sollte sich immer davon überzeugen, daß die LGO-Programme genau dahin geladen werden, wo man sie erwartet.

Man sollte auch die CP/M-Dokumentation und die Dokumentation seiner speziellen CP/M-Implementation zu Rate ziehen, um zu sehen, wie CP/M den Programmbereich TPA (temporary program area) verwaltet.

Wenn man ein LGO-Programm über CP/M setzt, gibt es keine Probleme. Wenn man es jedoch unter den CCP am oberen Ende der TPA setzt, muß man sein Programm so schreiben, daß die Adresse bei 0006 so geändert wird, daß sie auf den Beginn des Programms zeigen, das wiederum einen Sprung zur Adresse im BDOS enthalten muß, die ursprünglich bei Adresse

0006 vorhanden war. Dies bewirkt, daß CP/M mit einer reduzierten TPA normal operiert, wobei das Programm intakt bleibt, sogar wenn normale Programme in der TPA ablaufen.

Man kann nicht ein permanentes Programm ans untere Ende der TPA laden und die LGO-Datei darf nicht überlagert werden, während sie als normales Programm in der TPA ausgeführt wird. LGO ist in Small-C geschrieben; daher liegt sein Stapel am oberen Ende der TPA. Um deshalb das obere Ende der TPA für die zu ladenden Programme zur Verfügung zu haben, stellt LGO seinen Stack hinter sich an die ersten 256 Bytes.

Wenn LGO ausgeführt wird, muß ein Programmname in der Befehlszeile angegeben werden. Wenn eine Erweiterung angegeben ist, muß sie LGO heißen.

LGO führt folgende Funktionen aus:

- o Öffnet die genannte Datei.
- o Stellt sicher, daß sie das Laden-und-Ausführen Format hat.
- o Liest die Anfangsparameter.
- o Lädt die angegeben Anzahl Bytes an die angegebene Adresse.
- o Wenn -G angegeben wurde, wird die Kontrolle an die Startadresse übergeben.

Beispiele

BEFEHL KOMMENTAR

`LGO DRIVER -G`
Lädt DRIVER.LGO an die geeignete Adresse und beginnt mit der Ausführung.

`LGO PROG -M`
Lädt PROG.LGO, zeigt die Ladeadresse, Größe und Startadresse auf dem Bildschitm an und kehrt dann ins CP/M zurück.

`LGO ABC`
Lädt ABC.LGO, und kehrt nach CP/M zurück

Fehlermeldungen

MELDUNG ERKLÄRUNG

`xxxxxxxx - Can't Open`
Die genannte Datei kann nicht geöffnet werden.

`Error Reading xxxxxxxx`
Ein Ein-/Ausgabe-Fehler ist bei Lesen der angegebenen Datei aufgetreten.

`Invalid Extension`
Eine in der Befehlszeile angegebene Datei hat eine ungültige Erweiterung.

`Invalid LGO Format`
Die genannte Datei hat kein korrktes Laden-und-Ausführen-Format.

`xxxxxxxx - Too Long`
Die Befehlszeile enthält einen Dateinamen, der zu lang ist.

LIB: Der Small-Mac-Bibliotheksverwalter

Bedienung

```
LIB -{DPTUX}[A] library [module...]
```

-D	Löscht die genannten Module
-P[A]	Druckt die genannten oder alle (-PA) Module
-T[A]	Druckt ein Verzeichnis der genannten oder (-TA) aller Module
-U	Genannte Module aktualisieren (hinzufügen/ersetzen)
-X[A]	Genannte oder alle (-XA) Module herauslösen

Beschreibung

LIB wird benutzt, um eine Bibliothek verschiebbarer Objektmodule zu verwalten. Jedes Modul hat genau das gleiche Format wie ein frei stehendes (REL-Datei). Jede Bibliothek besteht aus einer Verkettung von Objektmodulen in einer Datei, die die Erweiterung LIB hat. Eine Indexdatei (NDX) muß jede Bibliothek begleiten. Die Indexdatei enthält eine Reihe von Wortpaaren, von denen jedes die Adresse des ersten Bytes eines Moduls innerhalb der Bibliothek angibt. Das erste Wort zeigt auf den Block und das zweite in den Block. LNK und LIB benutzen diese Information, um direkt das nächste Bibliotheksmodul zu suchen. LIB verwaltet Bibliotheken in alphanumerischer Ordnung.

Wenn LIB aufgerufen wird, muß der erste Parameter ein Schalter sein, der anzeigt, welche Funktion ausgeführt werden soll. Dieser muß vom Namen der in Frage kommenden Bibliothek gefolgt werden. Die Bibliothek kann die Erweiterung LIB haben (wird als Standard angenommen), aber keinen anderen. Eine Laufwerksangabe kann angegeben werden, um anzuzeigen, wo die Bibliothek gefunden werden kann.

Immer wenn LIB eine Bibliothek anlegt oder ändert, wird auf dem gleichen Laufwerk eine neue Bibliothek mit der Erweiterung L$ und ein neuer Index mit Erweiterung N$ angelegt. Bei erfolgreichem Abschluß werden die ursprüngliche Bibliothek und der ursprünglicher Index gelöscht und die neuen Dateien mit permanenten Erweiterungen umbenannt.

Die Schalter -D und -U erfordern eine Liste von Modulnamen, mit denen gearbeitet werden soll. Die anderen Befehle können entweder mit einer Liste der Module oder mit allen Modulen arbeiten. Die Liste kann auf drei Wegen angegeben werden: in der Befehlszeile, dem Bibliotheksnamen folgend, durch Eingabe über die Konsole oder von einer Datei mit Namen. Wenn die Namen in der Befehlszeile stehen, werden sie als Liste ge-

nommen. Sonst erhält LIB die Namen von der Standardeingabe. Wenn die Standardeingabe auf eine Diskettendatei umgelenkt wurde, liest LIB die Datei und erhält einen Namen pro Zeile. Wenn jedoch die Eingabe nicht umgeleitet wurde, fordert LIB jeden Namen über die Tastatur an. Eine leere Eingabe (nur Carriage-Return) signalisiert das Ende der Eingabe.

Die Schalter -P, -T und -X arbeiten mit jedem Modul in der Bibliothek, wenn die Namensliste leer ist; wenn also keine Namen in der Befehlszeile stehen und bei der ersten Anforderung keine Eingabe gemacht wurde oder wenn die Datei zu der die Standardeingabe umgelenkt wurde, leer ist. Wenn man die Aufforderung vermeiden möchte und auch keine leere Datei angeben möchte, fügt man einfach den Buchstaben A an den Schalter hinzu, also -PA, -TA oder -XA. Namenslisten können in jeder Reihenfolge erscheinen.

Modulnamen dürfen nur sechs Zeichen lang sein. Ein Dateiname kann jedoch acht Zeichen lang sein (ohne Laufwerksangabe und Erweiterung). In der Namensliste können Namen mit bis zu acht Zeichen enthalten sein. Solche Namen werden auf sechs Zeichen abgeschnitten, wenn sie sich auf Bibliotheksmodule beziehen. Der Schalter -U benutzt jedoch alle acht Zeichen, wenn nach freien Modulen gesucht wird, die in eine Bibliothek kopiert werden soll. Der Schalter -X benutzt nur sechs Zeichen, wenn er freie Module anlegt. Wenn Namen abgeschnitten werden sollen, zeigt LIB sie auf dem Schirm und fragt, ob es weitermachen soll oder nicht.

Module löschen

Mit dem Schalter -D werden Module aus der existierenden Bibliothek gelöscht. Jedes zu löschende Modul muß in der Namensliste angegeben sein.

Bibliotheksmodule drucken

Durch -P druckt LIB den Inhalt ausgewählter Module. Angegebene Module werden mit Programm-/Modulnamen, Programmgröße, Einsprungspunkte und der Programmendekennung gedruckt. Um Platz zu sparen, wird der Objekttext nicht gezeigt. Man kann DREL benutzen, um sich den vollständigen Inhalt der Module anzusehen.

Ein Inhaltsverzeichnis drucken

Durch den Schalter -T druckt LIB ein Inhaltsverzeichnis, also eine Liste der Modulnamen. Dies ist eine achtspaltige Liste in alphabetischer Reihenfolge, von links nach rechts und oben nach unten.

Eine Bibliothek anlegen oder aktualisieren

Durch den Schalter -T aktualisiert LIB die genannten Module (hinzufügen und ersetzen). Für jeden angegebenen Namen wird ein Modul in die Bibliothek kopiert. Wenn ein Modul mit dem gleichen Namen schon in der alten Bibliothek vorhanden ist, wird es ersetzt.

Bevor mit der Aktualisierung fortgefahren wird, wird jede genannte Datei auf der Diskette gesucht. Wenn eine nicht gefunden wird, fragt LIB ob es weitermachen soll. Wenn die genannte Bibliothek nicht existiert, wird eine leere angelegt und das Verfahren geht normal weiter.

Bibliotheksmodule herauslösen

Mit dem Schalter -X (Extract) kopiert LIB die genannten Module aus der Bibliothek als alleinstehende Module auf die Diskette. Jede Datei wird auf dem Standardlaufwerk mit REL-Erweiterung angelegt.

Beispiele

BEFEHL KOMMENTAR

`LIB -U M ABC DEF HIJ`
Aktualisiert M.LIB (und M.NDX) mit ABC.REL, DEF.REL, und HIJ.REL.

`LIB -X MY`
Löst Module aus MY.LIB heraus. Die Eingabe der Modulnamen erfolgt über die Tastatur. Wenn die erste Antwort eine leere Eingabe ist (CR), werden alle Module extrahiert.

`LIB -D MY <ABC.LST`
Löscht in MY.LIB (und MY.NDX) alle in der Datei ABC.LST genannten Module.

`LIB -P M GETREL`
Druckt das Modul GETREL aus M.LIB.

`LIB -TA C`
Gibt ein vollständiges Inhaltsverzeichnis von C.LIB aus.

Normale Meldungen

MELDUNG ERKLÄRUNG

`Added xxxxxx`
Das angegebene Modul ist zur Bibliothek hinzugefügt worden.

`Created xxxxxx`
Die angegebene Datei ist als freies Modul angelegt worden.

`Creating New Library`
: Die angegebene Datei existiert nicht, also wird eine mit diesem Namen angelegt.

`Continue?`
: Soll LIB weitermachen oder aufhören?

`Deleted xxxxxx`
: Das angegebene Modul ist aus der Bibliothek entfernt worden.

`Module Name: xxxxxx`
: Aufforderung für den nächsten Modulnamen.

`Replaced xxxxxx`
: Das angegebene Modul ist in der Bibliothek ersetzt worden.

Fehlermeldungen

MELDUNG ERKLÄRUNG

`xxxxxxxx - Can't Find - Ignored`
: Die genannte Datei kann nicht gefunden werden und wird ignoriert.

`xxxxxxxx - Can't Open`
: Die genannte Datei kann nicht geöffnet werden.

`Can't Rename Files`
: Die angeforderte Operation ist komplett, aber die Dateien können nicht in ihre permanten Erweiterungen umgewandelt werden.

`Close Error`
: Eine Datei kann nicht richtig geschlossen werden.

`Corrupt Library or Index`
: Eine Indexsuche konnte den Beginn eines Moduls nicht finden.

`Delete by Name Only`
: Die Module zum Löschen wurden nicht benannt.

`xxxxxxxx - Duplicate Name - Ignored`
: Das gleiche Modul wurde mehr als einmal angegeben.

`Error Reading Index`
: Ein Ein-/Ausgabe-Fehler ist aufgetreten, während die neue Indexdatei gelesen wurde.

`Error Writing New Index`
: Ein Ein-/Ausgabe-Fehler ist aufgetreten, während die neue Indexdatei geschrieben wurde. Wahrscheinlich ist die Diskette voll.

`xxxxxxxx - Extension Forced to xxx`
: Eine Dateierweiterung wurde angegeben. LIB ignoriert sie und verwendet stattdessen REL

`Invalid Extension`
: Eine Datei in der Befehlszeile enthält eine ungültige Erweiterung.

`xxxxxxxx - Invalid Format - Ignored`
: Es wurde ein ungültiges Dateinamensformat angegeben. Es wird ignoriert.

`Limited Stack Space`
: LIB arbeitet mit begrenztem Stapelplatz. Fehler sind möglich. Es wird eine größere TPA gebraucht.

`Memory Overflow`
: LIB kann nicht fortfahren, weil es nicht genug Speicher hat.

`Premature End of Index`
: Das Ende der Indexdatei wurde vor dem Ende der Bibliothek erreicht.

`Too Many Modules Specified`
: LIB kann die Anzahl der angegeben Modulnamen nicht verarbeiten. LIB kann höchstens 200 Modulanmen aufnehmen. Es können mehrere Bibliotheken angelegt werden oder LIB kann erneut mit einem größeren Wert für MAXMODS kompiliert werden.

`xxxxxx Was Not in Library`
: Das angegeben Modul wurde in der Bibliothek nicht gefunden.

`xxxxxxxx - Will be Truncated to xxxxxx`
: Der angegebene Dateiname wird, wie gezeigt, abgeschnitten.

`xxxxxxxx - Too Long`
: Die Befehlszeile enthält einen Dateinamen, der zu lang ist.

CMIT: Die Small-Mac-Anpassungsutility

Bedienung

```
CMIT [-C] [-L] [table] [mac]
```

`-C`	Passt den ausführbaren Assembler an die angegebene oder an die Standard-Maschineninstruktionstabelle (8080.MIT) an.
`-L`	Die kompilierte Maschineninstruktionstabelle listen.
`table`	Der Name der Maschineninstruktionstabelle im Quellformat.
`mac`	Name der Assembler-COM-Datei (Standard MAC.COM).

Beschreibung

CMIT wird benutzt, um die Maschineninstruktionstabelle vom externen Quellformat in internes Format zu kompilieren. In Anhang D sind die beiden Maschineninstruktionstabellen abgedruckt, die Small-Mac beinhaltet.

Wenn einmal eine Tabelle kompiliert worden ist, wird sie gedruckt und/ oder in den ausführbaren Assembler kopiert und paßt damit Small-Mac an eine spezifische CPU an.

Nachdem man einen neuen MAC.COM kompiliert und gelinkt hat, muß man ihn konfigurieren, indem man CMIT laufen läßt, bevor man ihn ausführen kann. Man kann einen schon konfigurierten MAC.COM jederzeit neu konfigurieren.

CMIT erzeugt seine Listings aus der Objekttabelle. CMIT liest die Quelltabelle ein zweites Mal und sucht jede Anweisung in der internen MI-Tabelle, wobei dieselben Funktionen wie beim Assembler benutzt werden. CMIT listet dann jede Anweisung, zeigt die Quelle, die Anzahl der Versuche um sie in der internen MI-Tabelle zu finden und den Objektcode, der erzeugt wird, wenn die Anweisung assembliert wird.

Wenn eine neue MI-Tabelle angelegt wird, muß man sorgfältig das Listing prüfen, ob es auch den korrekten Objektcode erzeugt.

Der Schalter -C

Mit dem Schalter -C konfiguriert CMIT den ausführbaren Assembler mit der kompilierten MI-Tabelle.

Der Schalter -L

Der Schalter -L bewirkt, daß CMIT die kompilierte Tabelle auf der Standardausgabe anzeigt. Die Ausgabe kann daher zu einem Drucker, zu einer Datei oder anderwohin umgelenkt werden.

Wenn keine Schalter angegeben sind, wird -L angenommen. Wenn jedoch irgendein Schalter vorhanden ist, wird nur die angeforderte Aktion durchgeführt.

Benennung der Maschineninstruktionstabelle

Wenn keine MIT-Quelldatei angegeben wird, wird 8080.MIT (auf dem Standardlaufwerk) angenommen. Ein Dateiname ohne Erweiterung oder mit Erweitwerung MIT bestimmt eine andere MIT-Quelldatei. Ein Laufwerk kann angegeben werden.

Benennung des Zielassemblers

Wenn kein ausführbarer Assembler genannt wird, wird MAC.COM auf dem Standardlaufwerk angenommen. Ein Dateiname mit der Erweiterung COM bestimmt eine andere Kopie des Assemblers. Ein Laufwerk kann angegeben werden.

Beispiele

BEFEHL KOMMENTAR

`CMIT`

Kompiliert 8080.MIT (vom aktuellen Laufwerk) und listet die sich ergebende Tabelle.

`CMIT -C`

Kompiliert 8080.MIT (vom aktuellen Laufwerk) und bringt eine Kopie nach MAC.COM (auch auf dem aktuellen Laufwerk).

`CMIT Z80 B:MAC.COM`

Kompiliert Z80.MIT (vom aktuellen Laufwerk) und bringt eine Kopie nach B:MAC.COM.

Normale Meldungen

MELDUNG ERKLÄRUNG

`Buffer Space Used nnnnn`

Die angezeigte Anzahl Bytes wurde als Puffer für die interne MI-Tabelle benutzt.

`Operation Codes nnnnn`
Die angezeigt Zahl von eindeutigen Operationscodes (Mnemonics) wurden kompiliert.

Fehlermeldungen

MELDUNG ERKLÄRUNG

`xxxxxxxx - Write Error`
Ein Ein-/Ausgabe-Fehler ist während es Schreibens auf die angegebene Datei aufgetreten. Wahrscheinlich ist die Diskette voll.

`Bad Expression Specifier`
In der Quelldatei wurde ein ungüliger Ausdruck gefunden.

`Bad Hex Byte`
In der Quelldatei wurde ein ungüliges hexadezimales Byte gefunden.

`Can't Find Instruction in MIT`
CMIT konnte bei der Überprüfung der Objekttabelle keine Anweisungsmnenomik finden. Wahrscheinlich ist dies einen Fehler in der MIT.

`Can't Find Operand`
CMIT konnte bei der Überprüfung der Objekttabelle keine Anweisungsoperanden. Wahrscheinlich wurden in der Quelldatei Anweisungen mit den gleichen Mnemoniks getrennt.

`xxxxxxxx - Can't Open`
Die genannte Datei konnte nicht geöffnet werden.

`Can't Rewind MIT File`
Die Quelldatei konnte nicht auf den Anfang positioniert werden.

`Close Error`
Eine Datei konnte nicht richtig geschlossen werden.

`Invalid Extension`
Eine Datei in der Befehlszeile enthält eine ungültige Erweiterung.

`MIT Buffer Overflow`
Dem internen MIT-Puffer wurde unzureichender Platz zugewiesen. Dies kann durch Erhöhung des Wertes für MIBUFSZ in MAC.H, mit anschließender Neukompilierung von CMIT und MAC behoben werden.

`xxxxxxxx MIT is nnnnn Bytes but should be nnnnn`
Die Größe der internen MIT im angezeigten ausführbaren Assembler stimmt mit der Größe in CMIT.COM nicht überein. Dies kann korrigiert werden, indem man überprüft, ob

MAC.H die richtigen Werte für MICOUNT, MIBUFSZ, und MIOPNDS enthält. Anschließend Neukompilierung von CMIT und MAC.

`MIT Mnemonic Overflow`

Der internen MIT für die Hashmnemonics wurde nicht genügend Platz zugewiesen. Dies kann durch Erhöhung des Wertes für MICOUNT in MAC.H und anschließender Neukompilierung von CMIT und MAC behoben werden.

`MIT Operand Overflow`

Der internen MIT für Operanden wurde nicht genügend Platz zugewiesen. Dies kann durch Erhöhung des Wertes für MIOPNDS in MAC.H und mit anschließender Neukompilierung von CMIT und MAC behoben werden.

`xxxxxxxx - Too Long`

Die Befehlszeile enthält einen Dateinamen, der zu lang ist.

DREL: Dump relokatierbarer Objektdateien

Bedienung

DREL

Beschreibung

DREL erzeugt ein formatiertes Listing vom Inhalt einer Objektdatei. Ausgegeben werden entweder einzelne Module oder Bibliotheken. Die Ausgabe geht zur Standardausgabe und kann daher zum Drucker, zu einer Datei oder anderswohin umgelenkt werden.

Schalter in der Befehlszeile werden nicht akzeptiert. Man wird nach jeder auszugebenden Datei gefragt. Wenn die Datei nicht gefunden wird, wird man weitergefragt. Dateinamen müssen mit Erweiterung angegeben werden. Laufwerksangaben sind zugelassen. Eine leere Eingabe beendet DREL.

Beispiel

```
library/module name: TEST.REL

-  program: TEST
 prog size: 008E'
   load at: 0064'
0064 0085' 05 00 EB CE 05 88 89 CD 0085' CD 008A' C3
0074 10 00 21 0089' 21 10 00 3A 00 00 3A 0005+ 007D'
0082 3A FFFB+ 0080' 01 32 03 34 05 36 07 38 09
 ext chain: 0083' EREF
- end prog: 0000
- end file

library/module name:
```

Die erste Spalte des Moduls zeigt die programmrelative Adresse des ersten Bytes, das rechts gezeigt wird.

Werte ohne Zusatz sind absolut. Werte mit Hochkomma (') sind programmrelativ. Werte mit einem Pluszeichen (+) sind Offsets, die LNK zur folgenden externen Referenz addiert, nachdem sie aufgelöst worden ist. Daher nehmen diese Werte im Programm keinen Platz ein und der Positionszähler wird durch sie auch nicht erhöht. Dies muß man beachten, wenn man Werte in einem Dump sucht. EREF ist eine externe Referenz mit dem Kopf der Kette bei 0083 hex. Versuchen Sie der Kette zu folgen.

Der Wert "load at" wurde durch einen Pseudo-Op ORG oder DS erzeugt. Der absolute Wert von 0000 bei "end prog" zeigt, daß keine Startadresse angegeben wurde.

5 Small-Tools

Brian W. Kernighan und P.J. Plauger haben in dem Buch "Software Tools" eine Philosophie beschrieben, in der Programme als Werkzeuge zur Problemlösung betrachtet werden. Jedes Programm oder Werkzeug ist dazu bestimmt, mit anderen Werkzeugen zusammenzuarbeiten. Jedes führt eine der Funktionen aus, die für eine vollständige Lösung des Problems gebraucht werden. So ein Werkzeugsatz kann auf verschiedene Arten kombiniert werden, um dann damit die gewünschten Ergebnisse zu erzielen.

Der Schlüssel, um mit diesem Konzept arbeiten zu können, liegt darin, daß die Ausgabe jeder Funktion kompatibel zur Eingabe jeder Funktion sein muß. Die Werkzeuge müssen also flexibel sein; sie dürfen nicht zu viele Vermutungen über die durchzuführenden Funktionen anstellen. Sie sollten etwas vernünftiges machen, sogar wenn ihre Parameter ungewöhnlich sind, da auch ungewöhnliche Effekte nützlich sein können.

Der Hauptvorteil dieses Ansatzes für den Programmentwurf ist, daß jedesmal wenn etwas benötigt wird, was sich nur gering von dem unterscheidet, was schon vorhanden ist, weder ein neues Programm entwickelt zu werden braucht, noch ein altes geändert werden muß. Vertrautheit mit den Funktionen und etwas Phantasie kann oft eine Lösung für ein neues Problem bringen.

Das Small-Tools-Paket

Small-Tools ist eine Programmsammlung, die durch das Buch "Software Tools" inspiriert wurde. Die Programme wurden besonders zur Verwendung auf Ein-Platz-Mikrocomputersystemen entwickelt.

Ihr Anwendungsbereich ist Textverarbeitung - ein Bereich mit hoher Aktivität, buchstäblich in jeder Zeile. Die Aufgaben, für die sie geeignet sind (auf die sie aber nicht begrenzt sind) umfassen:

- Schreiben von Briefen, Berichten, Artikeln und Büchern,
- Prüfung auf Schreibfehler (englisch),
- Schreiben von Computerprogrammen,
- Anlegen von Formularen und Dokumenten durch Beantwortung von Fragen,
- Zusammenstellung sinnvoller Dokumente aus vorher geschriebenem Material und/oder Beantwortung von Fragen,
- Adressdateien warten,

- o Druck individueller Serienbriefe entweder einzeln oder über eine Adressdatei,
- o Briefumschläge und Etiketten bedrucken.

Viele andere Möglichkeiten sind denkbar, abhängig von Notwendigkeit und Phantasie.

Die Small-Tools-Programme

Das Small-Tools-Paket besteht aus Programmen, die die folgenden Aufgaben mit Textdateien durchführen können:

- o Editieren
- o Formatieren
- o Sortieren
- o Zusammenfügen
- o Listen
- o Drucken
- o Suchen
- o Ersetzen
- o Übersetzen
- o Kopieren und Aneinanderfügen
- o Verschlüsseln und entschlüsseln
- o Leerzeichen durch Tabs ersetzen
- o Tabs durch Leerzeichen ersetzen
- o Zeichen, Wörter oder Zeilen zählen
- o Druckerzeichensatz wählen

Alle Programme arbeiten mit Dateien vom selben Format und man kann die Ausgabe eines Programms als Eingabe für ein anderes benutzen. Alternativ kann man die Ausgabe auch zur Konsole, zum Drucker oder zu jedem am Computer angeschlossenen Gerät schicken. Genauso kann natürlich auch die Eingabe von der Konsole oder von einem mit dem Computer verbundenen Gerät kommen.

Die Eingabe kann auch aus Disketten-Verzeichnissen stammen. Die besonderen Dateinamen A:, B: und so weiter stellen die Verzeichnisse auf den entsprechenden Laufwerken dar (X: ist das Standard-Laufwerk). Ein Verzeichnis sieht wie eine ASCII-Datei mit Dateinamen aus, einer pro Zeile. Diese Möglichkeit macht es einfach, Dateinamen zu wählen, um SUBMIT-Dateien für Operationen mit mehreren Dateien aufzubauen.

Kombiniert können diese Programme eine Vielzahl von Aufgaben durchführen.

Systemanforderungen

Diese Implementation des Small-Tools-Paketes läuft auf 8080/8085/Z80-Maschinen mit dem Betriebssystem CP/M, einem Diskettenlaufwerke und 56 KByte Speicher. Da diese Programme in Small-C geschrieben wurden und im Quellcode abgegeben werden, müssen sie vor der Verwendung mit dem Small-C-Compiler kompiliert, mit dem Small-Mac-Makroassembler assembliert und mit LNK gelinkt werden (näheres dazu in Kapitel 6).

Small-Tools, Konzepte und Möglichkeiten

Dateiformat

Jede Small-Tools-Datei hat das gleiche Format. Eine Datei aus einer Folge von Zeilen. Wenn sich die Datei auf Diskette befindet, folgt der letzten Zeile ein Dateiende-Byte mit dem Wert 26 dezimal oder 1A hex. In Fällen, wo die Datei mit dem letzten Byte des Sektors endet, wird das Dateiende-Byte nicht geschrieben. Wenn die Datei einem Ein-/Ausgabegerät zugewiesen ist, wird auch kein Dateiende-Byte geschrieben.

Eine Zeile besteht aus einer Folge von keinem oder mehr Zeichen, die durch zwei Zeichen, Carriage-Return (Wagenrücklauf) und Line-Feed (Zeilenvorschub) abgeschlossen werden. Eine Zeile kann höchstens 192 Zeichen (außer den beiden eben erwähnten) enthalten. Wenn man die Daten über die Tastatur eingibt, wird die Zeile automatisch nach dem 192. Zeichen beendet. Wenn Textdateien gelesen werden, unterbrechen die meisten Programme nach dem 192. Zeichen und beginnen die nächste Zeile mit dem folgenden Zeichen.

Wenn die Ausgabe auf eine Datei erfolgt, die schon existiert, wird sie überschrieben. Der Texteditor ist eine Ausnahme; zuerst nennt er die Datei in eine Datei mit der Erweiterung $$$ um und löscht nach dem erfolgreichen Schreiben die $$$-Datei.

Die meisten Programme haben eine Eingabe- und eine Ausgabedatei. Diese Standard-Dateien können zur Diskette oder zu Geräten umgeleitet werden. Ohne Umlenkung ist die Standardeingabe die Tastatur und die Standardausgabe dem Bildschirm zugeordnet. Man kann in der Befehlszeile Umlenkungsanweisungen angeben, um diese Standard-Zuweisungen zu ändern.

Die Umlenkungsanweisung für die Standardeingabe besteht aus dem Symbol "kleiner als" ("<"), sofort gefolgt von dem Dateinamen (im normalen CP/M-Format) oder einem logischen Gerätenamen (CON: oder RDR:) oder einem Inhaltsverzeichnis (A:, B:, ..., G:, X:). Die Zeichenkette <B:FILE3

leitet die Standardeingabe nach FILE3 auf Laufwerk B und <B: leitet dann weiter zum Verzeichnis auf Laufwerk B.

Die Umlenkungsanweisung für die Standardausgabe benutzt das Symbol "größer als" (">"). Dateinamen oder logische Geräte (CON:, LST: oder PUN:) können benutzt werden. Natürlich können Verzeichnisse nicht für die Ausgabe verwendet werden. Wenn die Standardausgabe durch ein Symbolpaar (zum Beispiel >>FILE3) umgelenkt wird, dann wird die Ausgabe an die schon vorhandenen Daten angehängt. Wenn die Datei noch nicht besteht, wird sie angelegt und ">>" unterscheidet sich nicht von ">".

Beide Umlenkungsanweisugen können gleichzeitig erscheinen und in jeder Reihenfolge und Position hinter dem Dateinamen in der Befehlszeile stehen. Man sollte Eingabe und Ausgabe nicht derselben Datei zuzuweisen; das Ergebnis wäre eine zerstörte Datei.

Format der Befehlszeile

Der Befehl zum Aufruf eines Small-Tools Programms besteht aus:

1. dem Programmnamen mit wahlweiser Laufwerkangabe im Standard-CP/M-Format,
2. Umlenkungsanweisung für Standardein- und -ausgabedateien,
3. Parameter zur Steuerung des Programms.

Ein oder mehrere Leerzeichen werden für die Trennung der Befehlsteile verwendet; daher kann ein Parameter keine Leerzeichen enthalten. Escapesequenzen (später beschrieben) können benutzt werden, um Leerzeichen in Parameter anzugeben.

Umlenkungsanweisungen werden von den Programmen nicht gesehen, deshalb können sie in jeder Reihenfolge nach dem Programmnamen erscheinen. Die Position der Parameter kann jedoch wichtig sein. Das CHG-Programm (Change) zum Beispiel braucht zwei Parameter, ein Textmuster, nach dem in der Eingabedatei gesucht werden soll und eine Zeichenkette, die diese Textmuster in der Ausgabedatei ersetzt. Der erste Parameter wird immer als Suchmuster und der zweite immer als zu ersetzende Zeichenkette interpretiert. Sie müssen in dieser Reihenfolge erscheinen.

Um effektiv zu sein, brauchen einige Programme Parameter in Groß- und Kleinbuchstaben. Das CP/M-Programm SUBMIT und der CCP wandeln jedoch die Kleinbuchstaben in große um, bevor sie an das Programm weitergegeben werden. Für die Erkennung von Kleinbuchstaben sind deshalb im Anhang A Patches für den CCP und SUBMIT beschrieben. Nach diesen Patches kann man mit den CP/M-Utilities Dateinamen in Kleinbuchstaben

angeben. Es ist zu empfehlen, Kleinbuchstaben nur bei Textmustern und zu ersetzenden Zeichenketten zu verwenden, da man bei CP/M-Dateinamen in Kleinbuchstaben auf Probleme stoßen kann. Die Small-Tools-Programme wandeln Dateinamen in Übereinstimmung mit CP/M-Konventionen immer in Großbuchstaben um.

Schalter in der Befehlszeile

Ein Sonderklasse von Befehlszeilenparametern, Schalter genannt, wird oft benutzt, um sekundäre oder Nebeneffekte zu kontrollieren. Schalter bestehen normalerweise aus einem Bindestrich, sofort von einem oder zwei Buchstaben gefolgt und in einigen Fällen von numerischen Werten. Der Schalter -BP123 zum Beispiel sagt dem Druckprogramm ab Seite (*P*age) 123 mit dem Ausdruck zu *b*eginnen. Wie die kursiven Zeichen anzeigen, soll man sich an Schalter leicht erinnern können.

Schalter können in der Befehlszeile in jeder Position hinter dem Dateinamen erscheinen. Nur die Schalter, die keine Parameter sind, sind positionsabhängig und das nur in gegenseitiger Beziehung. Schalter und Umlenkungsanweisungen können zwischen den anderen Parametern in jeder Folge erscheinen, ohne sie zu beeinflussen.

Da Schaltern immer der Bindestrich vorausgeht, sollte man Dateinamen vermeiden, die mit einem Bindestrich beginnen (diese Namen, die eigentlich keine Schalter sind, würden wie Schalter aussehen).

Bedienungshinweise

Damit man sich besser an die verschiedenen Schalter und Parameter erinnern kann, die in diesem Programm benutzt werden, wird jedesmal auf dem Bildschirm ein Bedienungshinweis angezeigt, wenn man Small-Tools-Programme ohne Schalterbuchstaben oder mit nicht definierten Schaltern aufruft. Wenn man also Hilfe braucht, braucht man nur den Programmnamen, gefolgt von einem Bindetrich, einzugeben. Der Bedienungshinweis zeigt die Syntax für den Programmaufruf; sie haben die Form:

```
usage: <Programm> <Parameter>... <Schalter>...
```

wobei <Programm> der Programmname, <Parameter> ein Parameter, <Schalter> ein Schalter ist und die Punkte anzeigen, daß einParameter mehrfach vorkommen kann. Schalter sind immer optional. Parameter können wahlweise sein oder nicht, je nach Programm. Umlenkungsanweisungen werden im Bedienungshinweis nicht gezeigt, da sie allgemein bei den Programmen verwendet werden können und ihre Verfügbarkeit angenommen werden kann.

Eckige Klammern ([]) erscheinen in den Bedienungshinweisen, um optionale Parameter anzuzeigen. Die Klammern sind nicht Teil des Befehls. Die Programme laufen mit Standardeinstellungen ab, wenn eine Angabe fehlt. Im Normalfall werden Schalter nur in Sonderfällen gebraucht.

Punkte (...) erscheinen in den Bedienungshinweisen, um anzuzeigen, daß ein Feld mehr als einmal vorkommen kann. Die Punkte selbst sind nicht Teil des Befehls.

Der senkrechte Strich (|) zeigt eine Alternative auf. Wenn man einen Bedienungshinweis liest, kann man den senkrechten Strich durch das Wort "oder" ersetzen. Genau wie die eckigen Klammern und die Punkte ist auch der senkrechte Strich nicht Teil des Befehls.

Das Nummernzeichen (#) steht für eine dezimale Integer mit einer oder mehreren Ziffern.

Das Fragezeichen (?) steht für ein Zeichen oder, in einigen Fällen, für eine Zeichenkette. Zahlen, Buchstaben und Sonderzeichen sind gültig.

Worte oder Wortkombinationen stehen für gültige Namen von besonderen Parametertypen. Der Term *outfile* zum Beispiel steht für den Namen einer Ausgabedatei, das Wort *pattern* steht für ein gültiges Suchmuster und so weiter.

Fehlerbehandlung

Alle Small-Tools-Programme, mit Ausnahme des Texteditors, behandeln Fehler auf die gleiche Art und Weise. Bei einem Fehler wird eine Meldung angezeigt, ein Warnton ausgegeben und das Programm beendet.

Besteht das Problem in einem ungültigen Kommando aus der Befehlszeile, wird der oben beschriebene Bedienungshinweis ausgegeben. Andere Fehler erzeugen andere geeignete Meldungen.

Alle Small-Tools-Programme haben zwei Fehlermeldungen gemeinsam. Die Meldung "output error" zeigt einen Fehler während des Schreibens einer Ausgabedatei auf die Diskette an. Höchstwahrscheinlich ist nicht genug Platz auf der Diskette. Da diese Fehlermeldung allen Programmen gemein ist, wird sie nicht unten bei den einzelnen Programmbeschreibungen aufgeführt. Die zweite allgemeine Fehlermeldung lautet: "<Datei>: can't open" . Dies bedeutet, daß auf der in Frage kommenden Diskette die Datei nicht gefunden werden kann.

Zwei Fehler werden vom Laufzeitsystem gefunden und nicht vom Programm selbst:

- R, Fehler bei der Umlenkungsanweisung, der einen Versuch anzeigt, die Standardeingabe auf eine nicht existierende Datei umzuleiten.
- M, Fehler bei der Speicherzuordnung, der einen Versuch anzeigt, mehr als den verfügbaren Speicher zuzuordnen. Dieser Fehler sollte niemals erscheinen.

Escapesequenzen

Manchmal ist es notwendig, nicht druckbare Zeichen oder Zeichen mit besonderer Bedeutung einzugeben. Man kann diese Zeichen über die Tastatur eingeben, wenn man den Doppelpunkt als Escape-Zeichen verwendet. Ein Escape-Zeichen ändert die Bedeutung des ihm folgenden Zeichens.

Vom Programm werden das Escape-Zeichen und das ihm folgende Zeichen als ein einziges Zeichen gesehen. Die Escape-Zeichen sind:

`:b`	Backspace
`:n`	neue Zeile (Carriage Return)
`:s`	Leerzeichen
`:t`	TAB
`:<Zchn>`	das angegebene Zeichen

Einige Zeichen, auch Metazeichen genannt, haben eine besondere Bedeutung, wenn sie in Suchmustern oder in zu ersetzenden Zeichenketten erscheinen. Man kann die Escapesequenz :<Zeichen> benutzen, damit sie als Zeichen selbst im Kontext gesehen werden.

Der Doppelpunkt ist ein Escape-Zeichen und muß zweimal angegeben werden, damit der Doppelpunkt selbst wirksam wird, also "::".

Ein Leerzeichen muß mit :s angegeben werden, da ein Leerzeichen in einem Parameter diesen normalerweise beendet.

Metazeichen

Einige Zeichen haben eine besondere Bedeutung wenn sie in Suchmustern oder zu ersetzenden Zeichenketten erscheinen. Als Gruppe bezeichnet man sie als Metazeichen (im Gegensatz zu den gewöhnlichen Zeichen). Da auch diese Metazeichen gelegentlich in einem Suchmuster oder in zu ersetzenden Zeichenketten erscheinen, müssen sie in diesen Fällen als Escape-Sequenz eingegeben werden. Alle Small-Tools-Programme benutzen die Definitionen der Metazeichen in TOOLS.H. Man kann die Zuweisungen nach Belieben ändern, indem man diese Datei vor dem Kompilieren ändert.

Hier eine Übersicht der Metazeichen:

Symbol/Name	**Verwendung**
:	Escape-Zeichen
'	Anfang einer Zeile
'	Ende einer Zeile
?	jedes Zeichen
*	kein oder mehrere Vorkommen eines Zeichens
[	Beginn einer Zeichenklasse
]	Ende einer Zeichenklassendefinition
-	kennzeichnet in einer Zeichenklasse einen Zeichenbereich
~	komplementiert eine Zeichenklasse
^	steht in einer zu ersetzenden Zeichenkette für die ganze ursprüngliche Zeichenkette

Suchmuster

Suchmuster werden vom Texteditor und einigen anderen Programmen benutzt, um ausgewählte Zeichenketten zu finden. Die einfachste Form eines Suchmusters ist eine Zeichenkette, die identisch zur gesuchten ist.

Wenn dem Suchmuster ein Akzent vorausgeht, bedeutet das, daß die Zeichenkette am Beginn einer Zeile erscheinen muß. An jeder anderen Position hat der Akzent keine besondere Bedeutung.

Schließt das Muster mit einem Apostroph ab, bedeutet dies, daß die Zeichenkette am Ende einer Zeile vorkommen muß. An jeder anderen Position hat der Apostroph keine besondere Bedeutung.

Die folgenden Muster zeigen die Benutzung der Metazeichen:

Muster	**Bedeutung**
'abcd	die Zeichenkette "abcd" am Anfang einer Zeile
xyz'	die Zeichenkette "xyz" am Ende einer Zeile
'xxx'	eine Zeile, die nur aus "xxx" besteht
ab'cd'e	die Zeichenkette "ab'cd'e", die irgendwo in der Zeile vorkommt

Ein Fragezeichen ("?") in einem Muster findet jedes Zeichen an dieser Stelle der Zeichenkette. Also findet das Muster f??t foot, feet, fit und so weiter.

Ein Stern "*" findet kein oder mehr Vorkommen eines Zeichens. Ein Stern am Anfang eines Musters hat keine besondere Bedeutung. Die folgenden Beispiele zeigen die Benutzung des Sterns.

Muster	gefundene Zeichenkette
`*abc`	*abc
`a*bc`	bc, abc, aabc, aaabc, ...
`aa*bc`	abc, aabc, aaabc, aaaabc, ...
`s?*p`	sp, sxp, sleep, s12 xp, ...
`the:s*man`	theman, the man, the man, ...

Man kann angeben, daß ein Zeichen an einer Stelle im Muster jedes aus einer Liste findet, aber keine anderen. Um dies zu erreichen, muß man nur die Liste der Zeichen in eckige Klammern einschließen. Zum Beispiel findet das Muster "ab[15Q]z" "ab1z", "ab5z", "abQz", aber nicht "ab3z".

Da die dezimalen Ziffern und die Klein- und Großbuchstaben oft benutzt werden und eine ziemlich lange Liste sind, existiert eine Abkürzung für "[012...9]", "[abc...z]", und "[ABC...Z]". Man kann zwischen dem ersten und letzten Zeichen einen Bindestrich stellen. Demnach findet das Muster "a[0-9]" "a0", "a1" und so weiter und das Muster "[A-Z][a-z]*" findet "A", "Able", "Zebra" und so weiter.

Man braucht also nicht alle Ziffern oder alle Buchstaben bei dieser Schreibweise anzugeben. "[5-7]", "a-g" reicht aus. Es gib nur die Einschränkung, daß die "kleineren" Zeichen vor dem Bindestrich stehen müssen. Man kann diese Schreibweise bei einer Liste von Zeichen, die selbst eine Zeichenklasse bilden, anwenden. Also ist "[s12g5-7a-zA-Z$(]" eine gültige Zeichenklasse.

Der Bindestrich "-" hat seine besondere Bedeutung nur, wenn er zwischen Zeichen in einer Zeichenklassendefinition steht. Wenn er an einem Ende der Definition oder außerhalb einer solchen Definition steht, hat er keine besondere Bedeutung.

Wenn das erste Zeichen nach einer linken eckigen Klammer eine Tilde ist ("~"), bewirkt das, daß jedes Zeichen gefunden wird, außer den aufgeführten.

Es ist wichtig eine Zeichenklasse immer als eine einzige Zeichenposition zu sehen.

Wenn man die Zeichen "[" und "]" in einem Muster braucht, kann man dies durch die Escapesequenzen ":[" bzw. ":]" erreichen.

Eingabe über die Tastatur

Jedesmal wenn man die Tastatur zur Eingabe benutzt, kann man Fehler vor dem Drücken der Return-Taste noch verbessern. Man kann Zeichen in umgekehrter Reihenfolge löschen; dazu muß man nur für jede Löschung Backspace (BS) oder DEL drücken. Für jede Löschung wird auf der Konsole die Folge "Backspace-Leerzeichen-Backspace" ausgegeben; wenn die Konsole ein Bildschirm ist, verschwindet das letzte Zeichen in der Zeile. Drucker gehen einfach eine Druckposition zurück. Die ganze Zeile kann man durch Control-X löschen.

Der CP/M-Druckerschalter (Control-P) arbeitet nicht mit Small-C-Programmen. Er ist auch im allgemeinen nicht erforderlich, da man die Programmausgabe auch auf LST: oder PUN: umlenken kann.

Man kann ein Small-Tools-Programm, das gerade ausgeführt wird, anhalten oder abbrechen. Ein angehaltenes Programm wartet einfach solange, bis die Ausführung wieder aufgenommen wird. Durch Eingabe von Control-S stoppt das Programm und durch noch eines (oder Control-Q) geht es weiter. Dies ist ganz praktisch, wenn die Ausgabe zu schnell auf dem Bildschirm erscheint. Man kann alternativ ein Programm anhalten und wieder weiterlaufen lassen, so daß man die sich die Ausgabe ansehen kann, bevor sie vom Schirm verschwindet. Ein laufendes oder angehaltenes Programm kann man durch Control-C abbrechen.

Jedesmal wenn die Standardeingabe eines Programms der Tastatur zugewiesen ist (normaler Zustand), fährt das Programm so lange mit der Verarbeitung fort, bis ein Control-Z eingegeben wird. Das Control-Z wird als Dateiendezeichen interpretiert. Es gibt keine Aufforderung an den Bediener, wenn das Programm auf eine Eingabe von der Konsole wartet und die Standardeingabe der Konsole zugewiesen wurde; das Programm wartet einfach auf zu verarbeitende Daten. Der Texteditor ist die einzige Ausnahme von dieser Regel.

Die Datei TOOLS.H

Während des Kompilierens wird die Datei TOOLS.H in jedes Small-Tools-Programm eingebunden. Sie definiert mehrere Systemparameter. Wie schon oben erwähnt die Metazeichen. Ebenso die maximale Größe der Textzeilen und die Dimensionen des Bildschirms und des Druckerpapiers. Man kann jeden dieser Werte ändern, um ihn an seine eigenen Bedürfnisse anzupassen. Auch sollte man sich diese Datei vor dem Kompilieren ansehen.

CHG (Change)

`CHG Muster [Ersatzzeichenkette]`

Beschreibung

CHG kopiert die Standardeingabe zur Standardausgabe. Bei der Verarbeitung wird der Text nach Mustern durchsucht; jedes gefundenen Vorkommen wird durch eine Zeichenkette ersetzt.

Das Suchmuster wird in der Befehlszeile eingegeben und wird gemäß den oben angegeben Regeln gebildet. Voller Gebrauch der Metazeichen und der Escapesequenzen ist erlaubt.

Die Ersatzzeichenkette ist eine beliebige Zeichenkette. Nur zwei Metazeichen haben in ihr Bedeutung: Circumflex, das für die ganze Zeichenkette steht, die mit dem Muster übereinstimmte und der Doppelpunkt, das Escape-Zeichen. Wenn man einen wirklichen Circumflex oder Doppelpunkt braucht, kann man Escapesequenzen verwenden.

Wenn keine Ersatzzeichenkette angegeben wird, wird eine leere Zeichenkette genommen, das heißt das gefundene Muster wird gelöscht.

Ein Suchmuster oder die Ersatzzeichenkette darf 48 Zeichen nicht überschreiten. Nur die ersten 48 Zeichen von längeren Zeichenketten werden verwendet.

Es ist gut, vor dem endgültigen Lauf, einen Versuchslauf mit Bildschirmausgabe zu machen. Wenn man sich den Effekt vorher ansieht, kann man überprüfen, ob man das Muster und/oder die Ersatzzeichenkette korrekt angegeben hat. Man sollte sich daran erinnern, daß man das Programm durch aufeinanderfolgende Control-S anhalten und weiterlaufen lassen und durch Control-C abbrechen kann.

Beispiele

`CHG <ABC eror error`

Kopiert die Datei ABC zur Konsole, ändert "eror" nach "error."

`CHG <F1 >F2 '[0-9]*`

Kopiert die Datei F1 nach Datei F2 und löscht alle führenden numerischen Ziffern aus jeder Zeile.

Meldungen

`pattern too long`

Die erweiterte interne Form des Musters ist zu groß, um in den reservierten Speicher zu passen.

`replacement too long`

Die erweiterte interne Form der Ersatzzeichenkette ist zu groß, um in den reservierten Speicher zu passen.

CNT (Count)

CNT [Datei] [-C|-W|-L]

Beschreibung

CNT durchsucht eine Eingabedatei und zählt alle Zeichen, Wörter und Zeilen. Die Ergebnisse werden auf der Standardausgabe angezeigt. Ein Parameter in der Befehlszeile, der kein Schalter ist, wird als Datei genommen, die für die Eingabe geöffnet wird. Wenn kein Dateiname angegeben wird, kommt die Eingabe von der Standardeingabe, die umgeleitet werden kann.

Wenn in der Befehlszeile kein Schalter angegeben ist, wird folgendes ausgegeben:

```
nnnnn characters
nnnnn words
nnnnn lines
```

wobei *nnnnn* eine Zahl zwischen 0 und 65535 ist.

Wenn einer der Schalter angegeben ist, gibt CNT eine einzige Zahl aus, entweder die Zahl der gefundenen Zeichen (-C), Wörter (-W), oder Zeilen (-L). Wenn mehr als ein Schalter vorhanden ist, wird nur der erste benutzt.

Die Zeichen Carriage-Return und Line-Feed, die jede Zeile abschließen, werden nicht mitgezählt. Um die Anzahl der Bytes in der Datei zu erhalten, muß man zur Zahl der Zeichen zweimal die Zahl der Zeilen plus ein Dateiendebyte addieren (wenn das Ergebnis nicht ein Vielfaches von 128 Bytes ist).

Ein Wort ist als zusammenhängende Zeichenkette druckbarer Zeichen definiert.

Beispiele

CNT <REPORT

Zeigt auf dem Bildschirm die Anzahl der Zeichen, Wörter und Zeilen in der Datei REPORT.

CNT <FILE >WORDS -W

Bringt die Zahl der Wörter in der Datei FILE in die Datei WORDS.

Meldungen

keine

CPY (Copy)

```
CPY [Datei]... [.?] [-B] [-NCR] [-NLF] [-T#,#]
```

Beschreibung

CPY ist ein Allzweck-Kopierprogramm. Es kopiert Standard-Small-Tools-Dateien genauso wie binäre Dateien, in denen der Dateiinhalt keinen Beschränkungen unterliegt. Wenn man mehr als eine Eingabedatei angibt, werden alle Dateien zu einer einzigen Ausgabedatei zusammengefügt. Man kann auch angeben, daß alle *#include*-Anweisungen (in C-Programmen) oder *.so*-Befehle für die Formatierung (in FMT-Textdateien) durch den Inhalt der genannten Dateien ersetzt werden.

Dateien, die kopiert werden sollen, werden in der Befehlszeile in der Reihenfolge aufgeführt, in der sie zusammengefügt werden sollen. Wenn keine Dateinamen vorhanden sind, wird die Standardeingabe genommen. Die Ausgabe erfolgt immer zur Standardausgabe.

Wenn *#include*- und *.so*-Dateien in die Ausgabe eingeschlossen werden sollen, dann muß der besondere Schalter .? in der Befehlszeile enthalten sein. Es ist üblich, Dateinamen mit zwei Teilen anzulegen, dem eigentlichen Namen und einer Erweiterung, die den Datentyp in der Datei beschreibt. Normalerweise werden diese Teile durch eine Punkt getrennt. Das Symbol "?" steht für eine Zeichenkette mit keinem oder mehreren der Erweiterung entsprechenden Zeichen, die von den am Kopierprozeß beteiligten Dateien benutzt werden. Wenn "?" leer ist (nur ein einzelner Punkt), dann werden alle entsprechenden Dateien eingeschlossen. Wenn "?" ein oder mehr Zeichen lang ist, dann werden nur die Dateien für die Kopie genommen, die mit der Erweiterung "?" übereinstimmen.

Der Schalter -B stellt binäre Kopie ein, also eine Byte-für-Byte-Kopie ohne eingeschlossene Dateien einzubeziehen und ohne bei einem Dateiende-Byte innerhalb der Dateien zu stoppen. Wenn die Eingabe über die Tastatur oder über ein I/O-Port kommt, beendet ein Control-Z die Eingabe. Wenn der Schalter ".?" bei einer binären Kopie angegeben wird, wird die Meldung "cannot include files during binary copy" ("Include-Dateien können bei binärer Kopie nicht verarbeitet werden") angezeigt und das Programm bricht ab.

Ohne den Schalter -B oder einen der Schalter für eine binäre Kopie, werden normale Textdateien unterstellt; in diesem Fall ist die Include-Funktion zugelassen und die Kopie einer Eingabedatei endet, wenn ein Dateiende-Byte gefunden wird.

Der Schalter -NCR bedeutet "no carriage return" und entfernt in der Eingabedatei alle Carriage-Return-Zeichen. Dieser Schalter setzt eine binäre Kopie voraus.

Der Schalter "-NLF" bedeutet "no line-feeds" und entfernt in der Eingabedatei alle Line-Feed-Zeichen. Dieser Schalter setzt ebenfalls eine binäre Kopie voraus.

Der Schalter -T#,# kann jedes gegebene Zeichen in einer Eingabedatei in ein anderes Zeichen übersetzen. Das erste Nummernzeichen steht für den dezimalen Wert des zu übersetzenden Zeichens; der zweite für den neuen Wert. Dieser Schalter setzt ebenfalls eine binäre Kopie voraus. Der Schalter wirkt nach den Schaltern -NCR und -NLF, also werden in Carriage-Return oder Line-Feed übersetzte Zeichen nicht umgesetzt.

Man kann diese drei letzten Schalter benutzen, um fremde Texte zu übersetzen.

Beispiel

`CPY ABC DEF >XYZ`

Schreibt den Inhalt von Datei ABC gefolgt von DEF zur Datei XYZ, und ersetzt alle *#include-* oder *.so-*Zeilen mit dem Inhalt der genannten Dateien.

Meldungen

`cannot include files`

Der *#include-*Schalter wurde zusammen mit einem Schalter für eine binäre Kopie angegeben.

CPT (Crypt)

`CPT Schlüssel`

Beschreibung

CPT wird benutzt, um Dateien jeden Typs zu ent- und verschlüsseln. Die Eingabe für CPT kommt von der Standardeingabe und die Ausgabe geht zur Standardausgabe.

Ein Begriff, im obigen Befehlsformat als "Schlüssel" bezeichnet, ist immer erforderlich. Er kann eine Zeichenkette mit einem oder mehreren Zeichen sein (maximal 80). CPT kombiniert den Begriff mit der Eingabedatei zyklisch mit einer Exklusiv-ODER-Funktion, um eine Ausgabedatei zu erzeugen. Die Verarbeitung nimmt keine Rücksicht auf den Zeichensatz oder das Dateiende-Byte in Textdateien. Daher kann man jede Dateiart verschlüsseln.

Wenn einmal eine Datei verschlüsselt wurde, kann man sie wieder entschlüsseln, indem man sie ein zweites Mal mit dem gleichen Schlüssel verarbeitet; also erhält man bei zwei Durchläufen mit dem gleichen Schlüssel wieder die Ausgangsdatei. Wenn man eine Datei mehrmals mit verschiedenen Schlüsseln unterschiedlicher Länge verarbeitet, macht das die Dechiffrierung schwieriger. Die Entschlüsselung ist nur eine Frage der Bearbeitung der verschlüsselten Datei mit den gleichen Schlüsseln wie bei der Verschlüsselung; die Reihenfolge der Schlüssel ist ohne Bedeutung.

Dieses Programm ist für die Benutzung von Dateien auf Disketten gedacht; es kann jedoch auch mit Daten von der Tastatur oder von einem anderen Ein-/Ausgabe-Gerät arbeiten, das an einem Ausgang des Computers angeschlossen ist. In diesen Fällen stoppt die Verarbeitung, wenn ein Control-Z gefunden wird. Das Control-Z erscheint nicht in der Ausgabe. Verschlüsselte Daten sind nicht druckbar; man erhält seltsame Ergebnisse auf dem Drucker oder dem Bildschirm.

Beispiele

`CPT <LIST >CLIST MAY`

Verschlüsselt die Datei LIST mit dem Schlüssel MAY zur Datei CLIST.

CPT <CLIST MAY

Wenn dieser Befehl dem obigen folgt, wird die Datei CLIST entschlüsselt und die Ausgabe erfolgt auf dem Bidlschirm.

Meldungen: keine

DTB (Detab)

```
DTB [#]... [+#]
```

Beschreibung

DTB wird benutzt, um Tab-Zeichen in einer normalen Textdatei durch die richtige Anzahl von Leerzeichen zu ersetzen, damit die Datei korrekt auf Geräten ausgegeben werden kann, die Tab-Zeichen nicht verarbeiten können. Die Eingabe erfolgt über die Standardeingabe, die Ausgabe zur Standardausgabe.

Wenn in der Befehlszeile keine Parameter vorhanden sind, werden Tabs an jeder achten Stelle angenommen, beginnend mit Spalte neun. Wenn diese Standard-Annahme nicht korrekt ist, kann man eine List von Zahlen in der Befehlszeile angeben. Jede Zahl gibt eine Spalte an, in der ein Tab existiert, damit die Eingabe normal auf einem Gerät ausgegeben werden kann, das keine Tabs kennt. Man kann der letzten Zahl ein plus Zeichen voranstellen, um anzugeben, daß ein Tab jede angegebene Spalte nach dem letzten Tab vorhanden sein muß. Wenn dem "+" keine Zahlen ohne Vorzeichen vorangehen, ist der erste Tab bei #+1.

Beispiel

```
DTB <SOURCE >LST: 5 +3
```

Kopiert die Datei SOURCE, die Tab-Zeichen für ein schreibmaschinenähnliches Gerät mit Tabs an Position 5, 8, 11 und so weiter enthält, zu einem Gerät, das dem logischen Gerät LST: zugewiesen ist.

Meldungen

```
tab stop beyond max line length
```

Es wurde versucht, ein Tab jenseits der maximalen Zeilenlänge zu definieren.

EDT (Edit)

EDT [Datei] [-V]

Beschreibung

EDT ist der Textditor für Small-Tools. Er wird benutzt, um normale Textdateien anzulegen und zu ändern. Wenn in der Befehlszeile eine Datei genannt wird, liest EDT diese Datei in den Editierpuffer ein und zeigt die ersten Zeilen auf dem Bildschirm. Der Editierpuffer ist leer, wenn keine Datei angegeben wird. Ein Nummernzeichen fordert einen Befehl an.

Wenn in der Befehlszeile der Schalter -v enthalten war, wird der V-Befehl vor allem anderen ausgeführt; damit wird nicht mehr automatisch der Puffer angezeigt. Dies ist wünschenswert, wenn man EDT in einer SUBMIT-Datei benutzt.

EDT erhält Befehle von der Standardeingabe. Es ist daher möglich eine Datei mit Editierbefehlen anzulegen, dann den Editor mit der zu dieser Datei umgeleiteten Standardeingabe aufzurufen. Dies ist bei Standard-Editierfunktionen praktisch, besonders wenn man den Editor aus einer SUBMIT-Datei aufruft.

Wenn EDT Zeilen aus dem Puffer anzeigt, wird die Ausgabe zur Standardausgabe geschickt. Man kann diese Ausgabe zu einer Datei oder einem anderen Gerät als dem Bildschirm umleiten. Es gibt jedoch keine vernünftigen Gründe, die Standardausgabe umzulenken.

Der Editierpuffer

EDT ist ein Editor, der im Speicher arbeitet; das bedeutet, daß die gesamte Datei in den Speicher des Computers passen muß. Das Editieren verändert nur die Kopie der Datei im Speicher; die Datei auf der Diskette bleibt unverändert. Durch einen Schreibbefehl wird der Pufferinhalt auf die Diskette geschrieben. Dabei wird entweder eine neue Datei angelegt oder eine vorhandene ersetzt.

Da das Programm für die Formatierung von Texten, das Kopierprogramm CPY und der Small-C-Compiler Include-Anweisungen unterstützen, kann man Dateien, die zu groß für den Speicher sind, in kleinere Teile untergliedern und sie dann als einzelne Dateien formatieren, kopieren und kompilieren.

Jedesmal wenn EDT eine Zeile im Puffer ändert, wird eine neue Version der Zeile angelegt. Die ursprüngliche Zeile wird logisch gelöscht, nimmt aber Platz im Puffer weg. Das gleiche gilt auch für gelöschte Zeilen. Aus

diesem Grund kann auch der Puffer überlaufen, wenn die Datei selbst zwar in den Puffer paßt, aber eine große Anzahl Änderungen gemacht wurde. In der Praxis passiert dies selten, wenn es aber doch einmal vorkommt, bewirkt ein Schreibbefehl eine Reorganisation des Puffers, wobei gelöschte Zeilen entfernt werden.

Zugriff auf Diskettendateien

Drei Befehle für die Übertragung von Texten zwischen Puffer und Dateien auf Diskette sind vorgesehen. Dies sind:

Befehl	Beschreibung
`e [Datei]`	Datei in Puffer einlesen
`r [Datei]`	Datei in Puffer dazulesen
`w [Datei]`	Datei aus dem Puffer schreiben

Der Enter-Befehl ersetzt den Inhalt des Puffers mit dem der Datei. Vom Enter-Befehl unterscheidet sich der Read-Befehl dadurch, daß die Datei an einem bestimmten Punkt in den Puffer geladen wird. Der Write-Befehl überträgt den Inhalt des Puffers zur Datei. Der gesamte Puffer oder ein Teil davon kann zur Datei geschickt werden, aber in jeden Fall enthält die Datei nur den tatsächlich geschriebenen Text.

Wenn man intensive Editieraufgaben ausführt, sollte man von Zeit zu Zeit den Write-Befehl verwenden, um den aktuellen Inhalt des Puffers auf der Diskette zu sichern. Wenn man dies nicht macht und der Computer einen Stromausfall hat oder man den Editor unbeabsichtigt verläßt, ist der Inhalt des Puffers verloren. Wenn man versucht, den Editor zu verlassen (Quit-Befehl) aber vergessen hat zur Diskette zu schreiben, erscheint die Warnung "didn't write to disk" ("nicht auf die Diskette geschrieben") und man wird zu einem anderen Befehl aufgefordert. Ein zweiter Versuch den Editor zu verlassen ist ohne Warnung möglich, auch wenn man vorher nicht zur Diskette geschrieben hat. Wenn im Puffer keine Veränderungen vorgenommen sind, wird keine Warnung ausgegeben.

Der Standard-Dateiname

Der Dateiname bei den Befehlen Enter, Read und Write ist wahlweise. EDT erinnert sich an die Datei, mit der gearbeitet wird. Bei jedem dieser Befehle ohne Dateiname erinnert sich EDT an die Datei, mit der gearbeitet wurde. Dies verringert die Fehlermöglichkeit durch wiederholte Benennung der Datei; ebenso wird auch die Zahl der Eingaben während des Editierens verringert.

Jedesmal wenn durch Enter eine Datei angegeben wird, wird dies der neue Standard-Dateiname. Die bei einem Read- oder Write-Befehl eingegeben Dateinamen werden nur dann Standard-Dateinamen, wenn keiner dieser drei Befehle seit dem Start des Programms eingegeben wurden.

Der Dateibefehl File (Datei) kann entweder zum Einstellen oder Anzeigen des Standard-Dateinamens benutzt werden. Wenn kein Name vorhanden ist, wird einfach der Standard-Dateiname angeziegt. Wenn ein Name da ist, wird dieser der neue Standard-Name.

Die aktuelle Zeile

Die aktuelle Zeile ist die Zeile im Puffer, die das Ziel des nächsten Befehls ist, wenn der Befehl ohne Zeilennummern angegeben wird. Gewöhnlich wird die letzte Zeile, in der ein Befehl ausgeführt wurde, die aktuelle Zeile für weitere Befehle. In einigen Fällen wird die veränderte erste Zeile die aktuelle Zeile werden. Die Beschreibung jedes Befehls sagt, wie die Position der aktuellen Zeilen verändert wird.

Die Kennzeichenspalte

Die am weitesten links stehende Spalte des Bildschirms wird nicht zum Anzeigen von Text benutzt sondern ist reserviert für die aktuelle Zeilenmarkierung, einen Stern und dem Befehlszeichen (Nummernzeichen). Durch die Zeilenmarkierung in der aktuellen Zeile weiß man über seine Position im Puffer Bescheid.

Zeilennummern

Die meisten Befehle arbeiten mit einer Zeile oder mehreren zusammenhängenden Zeilen. Um die Zielzeilen zu bestimmen, kann man eine oder mehrere Zeilennummern unmittelbar vor den Befehl setzen, wenn die Standard-Zeilennummern nicht geeignet sind. Bei Angabe von mehreren Zeilennummern muß man sie durch Komma oder Semikolon trennen. Bei der Benutzung eines Semikolons wird die erste Zahl zur aktuellen Zeile, bevor die zweite Zahl ausgewertet wird. Wenn man mehr Zeilen angibt, als vom Befehl gebraucht werden, werden die am weitesten rechts stehenden Zahlen benutzt.

Eine Zeilennummer entspricht immer der Position der entsprechenden Zeile im Puffer: Zeile 1 ist immer die erste Zeile im Puffer, Zeile 253 immer die 253. Zeile im Puffer. Dies bedeutet, daß sich die Zeilennummern während des Editierens ändern. Wenn zum Beispiel Zeile 25 gelöscht wird, wird aus der Zeile 26 die 25, aus 27 wird 26 und so weiter. Wenn eine Zeile vor der Zeile 5 eingefügt wird, dann wird aus der Zeile 5 Zeile 6, aus Zeile 6 wird

Zeile 7 und so weiter. Dies erweist sich jedoch nicht als schwierig, weil man Zeilennumern durch Sonderzeichen und Suchmuster symbolisch angeben kann. Wirkliche Zeilennummern werden nur selten angegeben.

Der Zeilenbefehl **L** zeigt die aktuelle Zeilennummer an, so daß man sie in Befehlen verwenden kann. Man kann auch einen Punkt (.) oder einen senkrechten Strich (|) als Zeilennummer angeben. Der Punkt steht für die aktuelle Zeile, der senkrechte Strich für die letzte Zeile im Puffer.

Eine Zeilennummer kann auch als durch Schrägstrich und umgekehrten Schrägstrich eingeschlossenes Suchmuster angegeben werden. Die Zeilennummer /abc/ ist die Nummer der ersten Zeile, die der Zeile folgt, die das Muster "abc" beinhaltet. Die Zeilennummer \'func()\ ist die Nummer der letzten Zeile, die der Zeile vorausgeht, die das Muster "func()" am Anfang der Zeile enthält.

Man kann Zeilennummern als Summe oder Differenz von Zahlen angeben; zum Beispiel gibt .+5 die fünfte Zeile nach der aktuellen Zeile an und der Ausdruck |-213 gibt die 213. Zeile vor der letzte Zeile an. Der Ausdruck /John Doe/-1 ist die Zeile, die der nächsten Zeile vorausgeht, die das Muster "John Doe" enthält. Jede Anzahl von Begriffen kann in einem Zeilenausdruck erscheinen. Wenn der Wert links vom Zeichen + oder - ausgelassen ist, wird "." angenommen. Wenn der Wert rechts fehlt, wird "1" angenommen. Also sind die folgenden Ausdrücke gültig:

`.+12-5`	entspricht `.+7`
`+5-2-11`	entspricht `.-8`
`.+7+`	entspricht `.+8`
`++++++++`	entspricht `.+8`
`-----`	entspricht `.-5`

Die folgenden Befehle zeigen die Benutzung von Zeilennummern:

`1,|p`	Jede Zeile im Puffer drucken.
`1,.d`	Jede Zeile vom Pufferanfang bis zur aktuellen löschen.
`\'1.\,/'12./p`	Alle Zeilen von der vorigen, die "1." am Anfang bis zur nächsten, die "12." am Ende enthält, drucken.
`.-2,.+2d`	Die aktuelle und zwei Zeilen davor und dahinter löschen.
`+23`	Die 23. Zeile nach der aktuellen wird die neue aktuelle.
`-----`	Die 5. Zeile vor der aktuellen wird die neue aktuelle.
`/abc/;.-1,.+1d`	Die nächste Zeile, die "abc" enthält finden, sie zur aktuellen machen, sie und eine Zeile davor und dahinter löschen.

Suchmuster

Befehle kombiniert mit Suchmustern können benutzt werden, um alle oben beschriebenen Möglichkeiten anzuwenden. Wenn Suchmuster als Zeilennummern verwendet werden, kann man nur die Begrenzung / und \ benutzen. Wenn sie im Ersetzungsbefehl oder bei den Parametern global oder ausgeschlossen erscheinen, wird das erste Zeichen nach dem Befehlsbuchstaben (s, g, or x) als Begrenzung benutzt.

Jedesmal wenn ein Suchmuster angegeben wird, wird es zum aktuellen Suchmuster. Auf das Standard-Muster wird durch zwei aufeinanderfolgende Begrenzungszeichen Bezug genommen ("\\" oder "//").

Editierbefehle

In der folgenden Befehlsbeschreibung sind die Zeilennummern in eckigen Klammern die Standard-Zeilennummern. Man kann sie durch andere Zeilennummern überschreiben. Wenn bei einem Befehl, der zwei Zeilennummern benötigt, nur eine angegeben wird, wird diese Zahl auch für die zweite genommen. Wenn mehr Zahlen vorhanden sind, als erforderlich sind, werden die am weitesten rechts stehenden genommen.

Der Begriff "angegebene Zeile" in der Befehlsbeschreibung bezieht sich auf die effektive Zeilennummer, ob Standard oder ausdrücklich gegeben, Symbol (. und |), Suchmuster (zum Beispiel /'abc/) oder aus diesen und den arithmetischen Operatoren (+ oder -) zusammengesetzt. Der Begriff "angegebene Datei" bezieht sich auf den effektiven Dateinamen, ob Standard oder ausdrücklich angegeben.

In den unten gezeigten Befehlsformaten sind die spitzen Klammern nicht Teil des Befehls; sie illustrieren nur die optionalen Befehlsteile. Das Symbol <text> steht für keine oder mehrere Textzeilen mit einem Punkt in Spalte eins. Das Nummernzeichen zeigt eine dezimale Integer an.

Die Befehlsbuchstaben können als Groß- oder Kleinbuchstaben geschrieben werden. Am Ende eines jeden Befehls kann das Druckbefehl (**P**) erscheinen. Dies ist ganz nützlich, wenn der Pufferinhalt nicht automatisch angezeigt wird und man die Auswirkung eines Befehls sehen möchte ohne einen zusätzlichen Druckbefehl zu geben.

Man kann jeden iterativen Befehl, außer Write, durch die Escape-Taste abbrechen. Des weiteren kann man die Druck- und Durchsuchbefehle durch jeden Tastendruck beenden. (Anhang E zeigt eine Zusammenstellung aller Editierbefehle.)

Beispiele

[.+1] — Dieser Befehl setzt einfach die aktuelle Zeile auf die angegebene. Wenn keine Zeilennummer vorhanden ist, dann wird die der aktuellen Zeile folgende Zeile zur neuen aktuellen. Durch Carriage Return bewegt man die aktuelle Zeile um eine nach unten - eine bequeme Möglichkeit um sich umzusehen.

[.]a
<text> — <text> nach der angegeben Zeile anfügen.
Wenn die Zeilennummer 0 ist, wird <text> vor die erste Zeile im Puffer gesetzt. Jedes Zeichen in <text> wird wie eingegeben genommen, Metazeichen und Escape-Sequenzen werden nicht erkannt. Zeichen können gelöscht werden, aber nur in der aktuellen Zeile. (BS oder DEL löschen das letzte Zeichen und ^X löscht die ganze Zeile.) Um vorhergehende Zeilen zu löschen, muß man den Append-Befehl beenden. Eine Zeile, die nur einen einzigen Punkt enthält beendet <text>, ist aber kein Teil von <text>.

[.,.]c
<text> — Ändert die angegeben Zeilen in <text>.
Nach Beendigung wird die erste Zeile von <text> die aktuelle. Eine Zeile, die nur einen einzigen Punkt enthält, beendet <text>, ist aber kein Teil von <text>.

[.,.]d — Löscht die angegebenen Zeilen. Die Zeile, die der letzten gelöschten folgt, wird die neue aktuelle. Wenn die letzte Zeile im Puffer gelöscht wird, wird die letzte übrigbleibende Zeile die aktuelle.

e [file] — Die angegebene Datei wird in den Puffer gelesen. Alles was vor dem Befehl im Puffer vorhanden war, ist verloren. Wenn im Puffer Änderungen gemacht worden sind, erscheint die Meldung "didn't write to disk" ("nicht auf Diskette geschrieben") und der Befehl wird ignoriert. Der zweite Versuch zum Einlesen ist immer erfolgreich. Die erste Zeile im Puffer wird zur neuen aktuellen.

f [file] — Einstellen oder Anzeigen des Standard-Dateinamens. Wenn ein Dateiname angegeben wird, wird er der neue Standard-Dateiname. In jedem Fall wird der Standard-Dateiname angezeigt. Dieser Befehl zeigt auch die Zahl der noch vorhandenen ungenutzten Bytes im Puffer an.

[.]i
<text> — <text> vor der angegebenen Zeile einfügen.
Die letzte Zeile von <text> wird die neue aktuelle Zeile. Einfügen unterscheidet sich vom Anfügen nur durch die Position von <text>. Eine Zeile, die nur einen einzigen Punkt enthält beendet <text>, ist aber kein Teil von <text>.

[.,.+1]j Die angegebenen Zeilen zu einer Zeile zusammenfügen. Die zusammengefügte Zeile wird die aktuelle.

l Die aktuelle Zeilennummer wird angezeigt.

[.,.]m# Verschiebt die angegeben Zeilen hinter die Zeile #. Die letzte verschobene Zeile wird die aktuelle.

[.,.]p[#] Zeigt die angegebenen Zeilen an. Wenn nur die aktuelle Zeile angegeben ist, dann wird auch der Kontext, der die angegebene Zeile umgibt, angezeigt. Der Kontext einer Zeile besteht aus # Zeilen über und unter ihr. Der Standard-Kontext ist 7. Jedesmal wenn # angegeben, wird das der neue Standard für zukünftige Ausgaben. Wenn die Konsole ein Bildschirm ist, wird dieser erst gelöscht. Die letzte ausgegebene Zeile wird die aktuelle. Wenn die Ausgabe durch Escape abgebrochen wird, wird die aktuelle Zeile vom Kontextwert abgefangen.

q Editieren beenden. Rückkehr ins Betriebsystem. Wenn im Puffer Änderungen vorgenommen waren, erscheint die Warnung "didn't write to disk" ("nicht auf Diskette geschrieben") und der Befehl wird ignoriert. Der zweite Versuch zum Verlassen des Editors ist immer erfolgreich.

[.]r [file]
Liest die angegebene Datei in den Puffer. Der neue Text wird hinter die angegebene Zeile gesetzt. Die letzte gelesene Zeile wird die neue aktuelle. Die Zeilennummer 0 kann angegeben werden, damit der Text vor der ersten Zeile im Puffer erscheint.

[.,.]s/pat/rep/[g]
Ersetzt *rep* durch das erste oder alle *pat* in den Zeilen. Wenn g (global) hinter dem Befehl erscheint, werden alle Suchmuster *pat* ersetzt; sonst wird nur das erste Suchmuster jeder Zeile ersetzt. Ein Circumflex (^) in *rep* ersetzt die ganze übereinstimmende Zeichenkette mit dem Suchmuster. Es kann beliebig oft vorkommen. Der Befehl "s/abc/^-^-^/" ergibt also "abc-abc-abc". Die Escapesequenz ":n" in der Ersatzzeichenkette teilt die Zeile bei jedem dieser Vorkommen in separate Zeilen auf. Die letzte geänderte Zeile wir die aktuelle.

v Zeigt automatisch die aktuelle Zeile im Kontext an. Standardmäßig wird bei jeder Änderung im Puffer oder beim Wechseln der aktuellen Zeile, die neue aktuelle Zeile im Kontext gezeigt. Diese Vorgehensweise kann durch v ein- und ausgeschaltet werden.

`[1,|]w [file]`
Schreibt die angegeben Zeilen zur angegebenen Datei, Wenn die Datei noch nicht existiert, wird sie angelegt. Wenn sie vorhanden ist, wird sie zuerst in eine mit der Erweiterung $$$ umbenannt und dann bei erfolgreichem Abschluß gelöscht. Wenn zufällig schon eine Datei mit $$$ existiert, wird sie erhalten und die ursprüngliche Datei wird direkt überschrieben. Nach erfolgreichem Abschluß wird die $$$-Datei gelöscht. Bei einem Schreibfehler ist wahrscheinlich nocht genug Platz auf der Diskette oder im Verzeichnis. In diesen Fällen wird eine Fehlermeldung ausgegeben, der Editor läuft weiter und die ursprüngliche Datei mit $$$-Erweiterung bleibt erhalten.

`[.,|]z` Den Editierpuffer anzeigen. Die angegebenen Zeilen werden solange am Bildschirm gezeigt, bis eine Taste gedrückt wird. Damit kann man sich schnell den Puffer ansehen.

Bei jedem der oben angegebenen Befehle außer bei Append (anfügen), Change (ändern), Insert (einfügen) und Quit (verlassen) können zwei Präfixe verwendet werden.

`[1,|]g/pat/command`
Das Präfix global (g) durchsucht die angegebenen Zeilen nach dem Suchmuster *pat*. Jede dieser Zeilen wird dann zur aktuellen und der Befehl (angegeben durch das Wort *command*) wird ausgeführt. Das Suchmuster im Präfix ist das vom Befehl verwndete Standard-Muster. Nach Beendigung ist die aktuelle Zeile die, in der die letzte Änderung vorgenommen wurde.

`[1,|]x/pat/command`
Das Außer-Präfix (x) arbeitet genau wie das Global-Präfix, nur das hier die Zeilen, die das Muster nicht enthalten, für die Verarbeitung ausgewählt werden.

Die Befehle, die mit dem Präfix *global* oder *außer* benutzt werden, können wie üblich ihre eigenen Zeilennummern haben; es ist jedoch kein Leerzeichen zwischen der am weitesten rechts stehenden Musterbegrenzung und dem Befehl erlaubt.

Meldungen

`didn't write to disk`
Es wurde versucht, eine Datei einzulesen oder den ganzen Puffer zu löschen, bevor die Änderungen im Puffer auf die Diskette geschrieben worden sind.

`error`

Ein Befehl wurde nicht richtig eingegeben oder eine Zeilennummer wurde als Suchkriterium eingegeben und die Suche blieb ohne Erfolg.

`memory overflow`

Der Editierpuffer im Speicher kann keine weiteren Zeilen mehr aufnehmen. Dies ist der Fall, wenn versucht wird, eine Datei zu lesen, die zu groß ist oder wenn zu viele Änderungen in der Datei gemacht wurden.

`open error`

Es wurde versucht, mit einem Read- oder Write-Befehl eine Datei zu öffnen, die auf der fraglichen Diskette nicht vorhanden ist.

`write error`

Während des Schreibens zur Diskette trat ein Fehler auf.

ETB

`ETB [#]... [+#]`

Beschreibung

ETB wird benutzt, um Leerzeichen in einer Standard-Textdatei durch Tabs zu ersetzen. Dies ist genau der umgekehrte Effekt von DTB.
Die Eingabe erfolgt über die Standardeingabe, die Ausgabe auf die Standardausgabe.
Wenn in der Befehlszeile keine Parameter angegeben sind, werden Tabs in jeder achten Spalte vermutet, beginnend mit Spalte neun. Wenn diese Standard-Annahme nicht korrekt ist, kann man eine Liste von Zahlen in der Befehlszeile angeben. Jede Zahl gibt eine Spalte an, in der ein Tab existiert. Man kann der letzten Zahl ein Pluszeichen voranstellen, um anzuzeigen, daß jede x Zeichen nach dem vorherigen Tab ein Tab ist. Wenn dem + keine Zahlen ohne Vorzeichen vorangehen, ist der erste Tab bei #+1.

Beispiele

ETB <ABC >DEF

Kopiert die Datei ABC nach DEF, ersetzt aufeinanderfolgende Leerzeichen durch entsprechende Tabs, wobei Tabs an jeder achten Stelle angenommen werden, beginnend mit Position neun.

ETB <XYZ >LST: 5 +3

Kopiert die Datei XYZ zum LST-Gerät, das Tabs an den Stellen 5, 8, 11 und so weiter hat.

Meldungen

`tab stop beyond 192`

Es wurde versucht, ein Tab hinter die maximale Zeilenlänge zu setzen.

FND

FND [~]Muster

Beschreibung

FND kopiert die Standardeingabe zur Standardausgabe. Dabei wird der Text nach Suchmustern durchsucht. Nur Zeilen die das Muster enthalten oder nicht enthalten gehen zur Ausgabe; FND wählt also aus einer Datei nur die Zeilen aus, die entweder das Suchmuster enthalten oder nicht enthalten.

Das Suchmuster wird in der Befehlszeile angegeben und wird nach den oben beschriebenen Regeln gebildet. Metazeichen und Escapesequenzen können vollständig benutzt werden.

Wenn dem Muster die Tilde (~) vorausgeht, dann werden die Zeilen ausgegeben, die das Muster nicht enthalten, sonst werden die Zeilen ausgewählt, die das Muster enthalten.

Beispiele

FND <ABC '[~:s:t]
: Zeigt auf dem Bildschirm alle Zeilen in der Datei ABC, die nicht mit einem Leerzeichen oder einem TAB beginnen.

FND <ABC >DEF ~''
: Kopiert die Datei ABC nach DEF und entfernt alle Leerzeilen.

Meldungen

pattern too long
: Die erweiterte interne Form des Musters ist zu groß, um in den reservierten Speicher zu passen.

FNT

`FNT [Gerät]`

Beschreibung

Dieses Programm wählt Schriftarten des Drucker aus. In der ausgelieferten Version wird aus den Schriftarten des Epson-FX-80 und kompatibler Drucker ausgewählt. Es ist jedoch ein einfaches Programm und kann leicht an andere Drucker angepaßt werden.

Ohne Datei (oder logischen Gerätenamen) in der Befehlszeile wird die Ausgabe zum LST-Gerät geschickt.

FNT zeigt das folgende Menü auf der Standardausgabe an (standardmäßig der Bildschirm) und wartet auf eine Eingabe von der Standardeingabe (standardmäßig die Tastatur):

ein	aus	Modus
1	2	Schmalschrift
3	4	doppelter Anschlag
5	6	Elite
7	8	hervorgehoben
9	10	Breitschrift
11	12	kursiv
13	14	Pica
15	16	tiefstellen
17	18	hochstellen
19	20	proportional

Eine gültige Antwort wird in die benötigte Control-Sequenz übersetzt und das Menü erscheint erneut zur weiteren Auswahl. Eine leere Antwort (RETURN) beendet das Programm. Es ist möglich jede Option individuell einzustellen und zu löschen.

Beispiel

FNT

Ausgabe zum LST-Gerät.

FNT PUN:

Ausgabe zum PUN-Gerät.

Meldungen: keine

FMT (Format)

```
FMT [Datendatei] [-BC#] [-EC#] [-BP#] [-EP#]
[-PO#] [-NP] [-NR] [-T] [-I] [-U] [-S] [-BS#]
```

Beschreibung

FMT ist der Textformatierer der Small-Tools. Textdateien mit eingeschlossenen Befehlen werden druckfertig formatiert. FMT erhält seine Eingabe primär von der Standardeingabe. Die Eingabe kann daher auch zur Diskette oder Tastatur geleitet werden. Die Ausgabe geht zur Standardausgabe, kann also zur Diskette, zum Bildschirm oder zum Drucker umgeleitet werden.

Man kann auch eine zweite Datei angeben, die Datendatei. In diesem Fall wird eine Kopie der primären Eingabedatei für jede Zeile der Datendatei erzeugt. Man kann jede Zeile der Datendatei in Felder aufteilen, indem man das Begrenzungszeichen (Standard "|") zwischen die Felder setzt, aber nicht am Ende einer Zeile. Man kann in der Primärdatei auf diese Felder Bezug nehmen, indem man eine Zahl zwischen zwei Begrenzungszeichen setzt. Wenn zum Beispiel |3| in der Primärdatei erscheint, bezieht sich das auf das dritte Feld in der aktuellen Zeile der Datendatei.

Wenn FMT einen solchen Verweis findet, wird er durch das angegebene Feld in der aktuellen Zeile der Datendatei ersetzt. Als Ergebnis erhält man eine individuelle Kopie für jede Zeile der Datendatei. Die offensichtliche Anwendung einer Datendatei ist die Erstellung von Serienbriefen, andere Anwendungen sind möglich.

Da es oft notwendig ist, nur einen Teil zu drucken, sind Schalter vorgesehen, die den Beginn und das Ende der Ausgabe kontrollieren. Der Schalter -BC12 bedeutet, mit der Kopie 12 zu beginnen; -EC25 bedeutet, mit der Kopie 25 aufzuhören. Kopie heißt in diesem Fall vollständige Kopien der Primärdatei, eine für jede Zeile in der Datendatei. Wenn diese Schalter ohne Datendatei angegeben werden, dann wird die angegebene Zahl der Kopien der Primärdatei erzeugt. In solchen Fällen ist der Schalter -BC# nutzlos und kann ausgelassen werden, da der Standard 1 ist. Das bewirkt, daß # im Schalter -EC# die tatsächliche Anzahl der Kopien angibt.

Der Schalter -BP2 bedeutet, mit der zweiten Seite zu beginnen; -EP112 bedeutet, auf Seite 112 aufzuhören. Seite heißt hier formatierte Seite innerhalb einer Kopie der Primärdatei. Wenn beide Schalter -BC13 und -BP2 vorhanden sind, beginnt der Druck auf Seite 2 mit Kopie 13 und geht dann für alle Seiten mit entsprechenden Kopien weiter. Wenn die Schalter -EC24 und -EP1 angegeben sind, endet der Druck nach der Seite 1 von Kopie 24.

Ein Seiteneinzug kann in der Befehlszeile angegeben werden. Der Schalter -PO5 gibt FMT an, alle ungeraden Seiten fünf Zeichen nach rechts und alle geraden Seiten fünf Zeichen nach links zu schieben. Damit kann man genug Platz für das Binden lassen, auch wenn das Papier beidseitig bedruckt ist.

Standardmäßig stoppt FMT vor jeder zu druckenden Seite. Angezeigt wird die Meldung "set page # ..." und FMT wartet dann auf eine Antwort, die angibt, daß Papier eingespannt und der Drucker bereit ist. Damit kann man also ein neues Blatt Papier einlegen. Mit Control-N von der Tastatur geht FMT direkt zur nächsten Seite. Auch dabei wird wieder die Meldung "set page # ..." vor dem Weiterdruck angezeigt. Jede andere Antwort bewirkt, daß FMT mit dem Druck beginnt.

Mit dem Schalter -NP druckt FMT ohne Pause; nach der ersten "ready printer ..."-Anzeige, wird ohne Pause zwischen den Seiten gedruckt. Diese Option wird bei Endlospapier verwendet. Mit dem Schalter -NR übergeht FMT die Ready-Meldung. Das ist nützlich, wenn FMT in einer SUBMIT-Datei ohne Unterbrechung verwendet wird.

Es gibt zwei Arten um Unterstreichung, Doppeldruck und Kursivdruck zu kontrollieren. Das ist einmal der Epson-Modus und zum anderen der TTY-Modus (nicht-intelligenter Drucker). Im Epson-Modus (standardmäßig) werden Steuerzeichensequenzen zum Drucker geschickt, damit mit einem Minimum an Kopfbewegungen unterstrichen, halbfett oder kursiv gedruckt werden kann. Drucken mit doppelter Breite ist auch möglich. Im TTY-Modus (spezifiziert durch den Schalter -T) wird durch mehrere Anschläge unterstrichen und halbfett gedruckt. Drucken in doppelter Breite oder kursiv ist überhaupt nicht möglich.

Mit dem Schalter -I erreicht man, daß unterstreichen als kursiv interpretiert wird.

Mit dem Schalter -U erreicht man, daß kursiv als unterstrichen interpretiert wird.

Mit dem Schalter -S erreicht man, daß FMT in der Ausgabe die Namen der Dateien anzeigt, die im Befehl ".so file" angegeben sind. Damit kann man die Dateigrenzen in längeren Ausdrucken besser identifizieren.

Im TTY-Modus gibt der Schalter -BS# die Zahl der Standardanschläge für Fettdruck an. # ist die Anzahl der zu verwendenden Anschläge.

Befehle

Befehle, die FMT sagen, was zu tun ist, werden im zu verarbeitenden Text eingeschlossen. Jede Zeile, die mit einem Punkt (oder einem anderen angegeben Zeichen) beginnt, wird als Befehl interpretiert. Jeder Befehl besteht aus zwei Buchstaben, die dem Punkt sofort folgen. Die Buchstaben sind nach mnemonischen Gesichtspunkten gewählt worden und können daher sehr leicht behalten werden.

Die meisten Befehle haben noch numerische oder Zeichenparameter in der gleichen Zeile. Ein oder mehrere Leerzeichen trennen die Befehle von ihren Werten. Befehle können in Klein- oder Großbuchstaben angegeben werden.

Anstelle eines Punktes kann das Befehlskennzeichnen auch ein anderes Zeichen sein, wenn man den Befehl .cc ? eingibt, wobei das ? das neue Befehlskennzeichen ist. Man kann den Befehl ?cc . eingeben, um wieder den Punkt als Befehlskennzeichen zu haben. Durch Änderung des Befehlsflags kann man Text mit führenden Punkten in den Zeilen verarbeiten.

Jeder Befehl, der einen numerischen Wert braucht, akzeptiert diesen absolut oder relativ. Wenn eine Zahl ohne Vorzeichen angegeben wird, wird sie als neuer Wert interpretiert. Wenn der Zahl ein Minus oder Plus vorausgeht, dann wird diese Zahl zum derzeitig wirksamen Wert addiert oder subtrahiert und man erhält dann den neuen Wert.

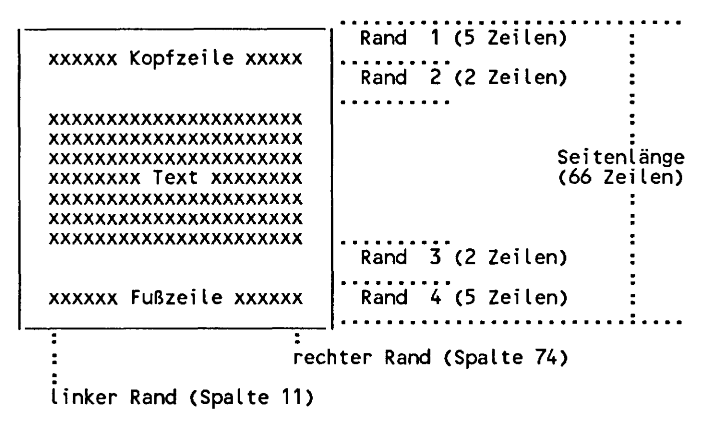
Wenn zum Beispiel ein Textabschnitt vier Stellen eingerückt und danach dann wieder normal weitergemacht werden soll, kann man angeben:

```
.in +4
(Text)
(Text)
(Text)
.in -4
```

Durch diese Anordnung kann man lokal Änderungen in der Formatierung machen ohne auf den Gesamttext Rücksicht nehmen zu müssen. Allgemeine Befehle kann man an den Anfang der Datei stellen, wo man sie leicht finden und ändern kann. Zum Beispiel sollte man nicht die ganze Datei nach jedem Vorkommen eines Befehls zu durchsuchen haben, wenn man eine vorhandene Datei mit anderen Randabständen drucken will.

Das Layout einer Seite

Die durch FMT erzeugten Seiten haben das in der folgenden Abbildung gezeigte Format.



Die Dimensionen einer Seite sind in Zeilen (horizontal) oder Spalten (vertikal) angegeben. Jede Angabe wird mit den Standardwerten in Klammern gezeigt.

Der Abstand am Seitenanfang und Seitenende ist jeweils 5 Zeilen breit. Dies ist der effektive Abstand; der tatsächliche Standardabstand für Rand 1 und 4 ist 1 bzw. 9. Die Ränder entsprechen dem Seitenformat, wenn das Papier so im Drucker eingespannt wird, daß die erste Druckzeile, Rand 1, etwa vier Zeilen vom Seitenanfang beginnt. Dies ist die normale Papierposition für den Seitenanfang in einem Drucker.

Wenn man Endlospapier benutzt, sollte man den Schalter -NP setzen. In diesem Fall wird keine Pause zwischen den Seiten gemacht. Der Rand am Ende der Seite überspringt die Perforation zwischen den Seiten und läßt am Anfang und am Ende einer Seite gleichviel Platz.

Wenn man Einzelblattpapier benutzt, muß man sie einzeln in den Drucker einspannen und sollte den Schalter -NP nicht angeben. In diesem Fall macht FMT eine Pause zwischen den Seiten und wartet auf das nächste Blatt. Die Pause erfolgt sofort nach Drucken der Fußzeile. Der Druck geht sofort mit der Kopfzeile der nächsten Seite weiter. Also liegt es in jedermanns eigener Verantwortung, dafür zu sorgen, daß die gedruckte Seite entfernt und die neue genau da eingespannt wird, wo weitergedruckt werden soll.

Mit den folgenden Befehlen kann man das Standard-Layout einer Seite ändern:

```
.m1 #      Rand 1
.m2 #      Rand 2
.m3 #      Rand 3
.m4 #      Rand 4
.pl #      Seitenlänge
.lm #      linker Rand
.rm #      rechter Rand
```

Diese Befehle können irgendwo im Text stehen und sie werden an dieser Stelle wirksam. Üblicherweise setzt man sie an den Dateianfang und läßt sie dann unverändert stehen.

Wenn man die Seitenlänge Null angibt (.pl 0), dann wird der Text als eine Seite von unendlicher Länge betrachtet. Dabei wird normal formatiert, es gibt nur keine Seitenunterbrechung und keine Fuß- und Kopfzeilen. Nützlich ist diese Formatierung zur Eingabe für andere Programme.

Kopf- und Fußzeilen

Kopf- und Fußzeilen sind optional. Sofern man sie nicht explizit angibt, erscheinen sie als Leerzeilen innerhalb von Rand 1 und Rand 4. Einmal angegeben erscheinen diese Zeilen automatisch an der richtigen Stelle auf jeder Seite. Dies bleibt bis zur Neudefinition wirksam. Fußzeilen beginnen auf der aktuellen, Kopfzeilen auf der folgenden Seite.

Nur eine Zeile wird für Kopf- oder Fußzeile benutzt. Die Kopfzeile erscheint als letzte Zeile von Rand 1, die Fußzeile als erste Zeile von Rand 4. Diese Ränder müssen daher wenigstens eine Zeile umfassen, damit Kopf- und Fußzeilen erscheinen können.

Der Befehl *.he [Text]* wird benutzt, um eine Kopfzeile mit dem Text zu definieren, der dem Befehl folgt. Der Befehl *.fo [Text]* definiert genauso eine Fußzeile. Wenn kein Text angegeben wird, erscheint eine leere Kopf- oder Fußzeile. Mit FMT kann man unterschiedliche Kopf- und Fußzeilen für gerade und ungerade Seiten definieren. Es gibt sechs Befehle:

```
.he [Text]  Kopfzeile
.oh [Text]  Kopfzeile auf ungerader Seite
.eh [Text]  Kopfzeile auf gerader Seite
.fo [Text]  Fußzeile
.of [Text]  Fußzeile auf ungerader Seite
.ef [Text]  Fußzeile auf gerader Seite
```

Es gibt zwei Zeichen mit besonderer Bedeutung innerhalb von Kopf- und Fußzeilen. Das Nummernzeichen # steht für die aktuelle Seitenzahl. Jedesmal wenn es in Kopf- oder Fußzeile erscheint, wird es durch die aktuelle Seitenzahl ersetzt. Den Schrägstrich / kann man verwenden, damit der Text gleichmäßig zwischen den Rändern verteilt erscheint. Die Schrägstriche werden so durch Leerzeichen ersetzt, daß das erste Zeichen auf dem linken Rand, das letzte Zeichen auf dem rechten Rand steht. Durch umsichtige Benutzung von Schrägstrichen kann man Text zentrieren, Text dehnen und Text einseitig ausrichten.

Durch den Befehl "fo /- # -/" erhält man eine zentrierte Fußzeile mit von Bindestrichen umgebener Seitenzahl.

Zeilen füllen und Blocksatz

Wenn nicht anders angegeben, gibt FMT keine Zeile im Text aus, so lange nicht genug Wörter da sind, um die Zeile zu füllen; Wörter werden also zuerst in einer Zeile gesammelt, bis zu dem Wort, daß den rechten Rand überschreiten würde. Dieses Wort bleibt gespeichert und erscheint als erstes in der nächsten Zeile. Dieses Verfahren wird als Zeilenfüllen bezeichnet. Dies kann durch den Befehel *.nf* (nicht füllen) abgeschaltet und durch den Befehl *.fi* (füllen) wieder eingeschaltet werden. Wenn FMT zusammenhängende Leerzeichen entdeckt, werden alle außer dem ersten ignoriert. Zwei Leerzeichen werden hinter den Punkt gesetzt, der einen Satz beendet.

FMT fügt automatisch Leerzeichen so zwischen Wörter in einer Zeile ein, daß das letzte Zeichen des letzten Wortes genau auf dem rechten Rand steht. Dies bezeichnet man als Blocksatz. Durch den Befehl *.nj* läßt sich dies ausschalten, durch *.ju* wieder einschalten.

Manchmal ist es wichtig, daß eine bestimmte Anzahl von Leerzeichen an einer gegebenen Stelle in einer Zeile erscheinen. Der Blocksatz und das Zeilenfüllen sollten nichts daran ändern. Für die zusätzliche Kontrolle dieser Umstände ist die Tilde (~) vorgesehen, die ein Pseudo-Leerzeichen darstellt. Es wird wie jedes andere Zeichen bei der Ausrichtung und beim Zeilenfüllen behandelt, wird jedoch vor der Ausgabe in ein Leerzeichen umgewandelt.

Wenn die Tilde als Tilde erscheinen muß, kann man ein anderes Zeichen als Pseudo-Leerzeichen definieren und zwar durch den Befehl *.bc x*. Dabei wird *x* jetzt das Pseudo-Leerzeichen und der Befehl *.bc* ~ gibt der Tilde ihren Ausgangsstatus zurück.

Zeilenunterbrechung

Unter gewissen Bedingungen während FMT Zeilen füllt, kann eine Zeile gedruckt werden, die noch nicht voll ist. Dies bezeichnet man als Zeilenunterbrechung. In diesen Fällen gibt es keinen Blocksatz, das heißt alle Leerzeichen erscheinen rechts der Zeile.

Eine Ursache für die Zeilenunterbrechung ist eine Leerzeile im Text. Wenn FMT auf eine Leerzeile trifft, wird die aktuelle Zeile so ausgegeben wie sie ist und dann die Leerzeile gedruckt. Dies ist eine einfache Möglichkeit Text in Absätze zu gliedern, getrennt durch zwei Leerzeilen.

Bestimmte Befehle bewirken eine Zeilenunterbrechung; dies sind:

`.fi`	füllen
`.nf`	nicht füllen
`.ju`	Blocksatz
`.nj`	kein Blocksatz
`.br`	Unterbrechung
`.sp #`	# Leerzeilen
`.bp #`	neue Seite
`.in #`	um # Stellen einrücken
`.lm #`	linken Rand auf # setzen
`.ti #`	zeitweise um # Stellen einrücken
`.ce #`	die nächsten # Zeilen zentrieren
`.sq #`	Ränder um # Stellen zusammendrücken
`.ne #`	# Zeilen zusammenhalten

Die letzte Ursache für eine Zeilenunterbrechung ist eine Zeile mit einem oder mehreren führenden Leerzeichen. Eine solche Zeile wird mit den Leerzeichen ausgegeben; wenn Zeilenfüllen und Blocksatz eingeschaltet sind, werden sie auf die Zeile angewendet.

Seitenunterbrechung

Während FMT Text verarbeitet, wird die Anzahl der Zeilen, die schon auf der Seite stehen, gespeichert. Wenn keine Zeilen mehr hineinpassen, werden Rand 3 und 4, möglicherweise mit einer Fußzeile, ausgegeben und eine neue Seite beginnt. Wenn FMT zwischen den Seiten stoppt (Normalfall) hört man den Lautsprecher und sieht die Meldung "set page # ..." wobei # die neue Seitenzahl angibt. In diesem Fall werden die der Fußzeile folgenden Zeilen von Rand 4 nicht gedruckt. Dies verhindert, daß die Kante am Ende der Seite über den Druckmechanismus schleift und sich vielleicht in die bewegten Teile verklemmt.

Bei fortlaufendem Druck beginnt FMT sofort ohne Unterbrechung mit der nächsten Seite.

Seitenunterbrechungen können an jeder Stelle im Text durch den Befehl *.bp* # bewirkt werden. Wenn im Befehl eine Zahl erscheint, wird diese Zahl die neue Seitenzahl; sonst erhält die neue Seite eine um eins größere Seitenzahl als die vorherige.

Die Seitenzahl hat drei Bedeutungen. Erstens kann sie in Kopf- und Fußzeilen gedruckt werden, zweitens gibt sie die Richtung des Seitenoffsets (*.po*) an. Drittens kann damit die Stelle im Text angegeben werden, an der der Druck beginnt oder endet.

Einrücken

Einrücken kann man, wenn man eine Anzahl Leerzeichen zum linken Rand hinzufügt oder von ihm entfernt. Der Standardwert ist Null. Man kann den Befehl *.in* # benutzen, um die Einrückung aller folgenden Zeilen festzulegen. Der Befehl *.ti* # wirkt nur temporär auf die folgende Zeile. Dies ist ganz praktisch für die Einrückung von Absätzen. In beiden Befehlen ist der Standardwert für # Null.

Eine Sonderform des Einrückens bewirkt der Befehl *.sq* #. Dabei rückt FMT von beiden Rändern um # Stellen nach innen. Dies ist sinnvoll, wenn man Zitate hervorheben will. Der Standard-Wert für # ist Null. Diese Druckform wird solange beibehalten, bis ein *.sq* oder *.sq 0* gefunden wird. Am linken Rand werden die aktuellen und die temporären Werte zum Wert des Befehls *.sq* addiert. Durch ein Pluszeichen kann der aktuelle Wert von *.sq* um # erhöht, durch ein Minuszeichen vermindert werden.

Zentrieren

Durch den Befehl *.ce* # werden die nächsten # Zeilen im Text zentriert. der Standardwert für # ist 1, also zentriert *.ce* nur die folgende Zeile. Der Befehl *.ce 0* stoppt die Zentrierung; dies ist praktisch, wenn eine unbestimmte Zeilenzahl zentriert werden soll. Zuerst gibt man *.ce 1000* an (oder irgendeine Zahl größer als benötigt, aber nicht größer als 32767), gefolgt von dem zu zentrierenden Text; am Ende erscheint der Befehl *.ce 0*.

Die Zentrierung erfolgt zwischen dem linken und rechten Rand; die Einrückungswerte wirken sich nicht aus.

Unterstreichen

Text kann durch einen der beiden vorhandenen Befehle unterstrichen werden. Durch *.ul* # werden die nächsten # Zeilen unterstrichen, Leerzeichen zwischen den Wörtern werden nicht unterstrichen. Mit dem Befehl *.cu* # wird fortlaufend unterstrichen, also auch die Wortzwischenräume. Sonderzeichen werden normal nicht unterstrichen; das fortlaufende Unterstreichen unterstreicht jedoch auch eckige, geschweifte, runde Klammern und Semikolons.

Die Befehle *.ul* und *.cu* schließen sich wechselseitig aus. Wenn einer gegeben wird, während der andere noch wirksam ist, wird dessen Wirkung ausgeschaltet.

Genau wie bei der Zentrierung können auch hier beliebig viele Zeilen durch einen entsprechend hohen Wert für *.ul* # oder *.cu* # unterstrichen werden. Nach dem Text kann man dann durch *.ul 0* oder *.cu 0* das Unterstreichen wieder beenden. Eine andere Möglichkeit der Beendigung erhält man durch den Befehl *.nu* (nicht unterstreichen). Jeder dieser drei Befehle kann jede Art des Unterstreichens beenden.

FMT arbeitet beim Unterstreichen, im Kursivdruck und Doppeldruck in einem von zwei Modi. Im Epson-Modus (Standard) wird dadurch unterstrichen, daß die korrekten Kontrollzeichen vor und hinter den zu unterstreichenden Zeichen erzeugt werden; der Drucker (Epson oder ein anderer Nadeldrucker) unterstreicht dann während des Druckens. Im Teletype-Modus (Schalter -t) erreicht man unterstreichen dadurch, daß jedem zu unterstreichenden Zeichen eine Sequenz aus Unterstrich und Backspace vorangestellt wird. Abhängig vom vorhandenen Druckmechanismus kann dieser Modus sehr langsam sein und man erhält eine abgehackte Hin- und Herbewegung des Druckkopfes.

Der Kommentar-Befehl

Der Befehl *.Text* bewirkt, daß der *Text* auf dem Bildschirm angezeigt wird, wenn das Programm ihn findet. Zuerst ertönt der Lautsprecher, dann erscheint auf dem Bildschirm das Wort *note*, gefolgt von dem entsprechenden Text. Diese Kommentarzeile wird bei der Formatierung nicht berücksichtigt. Man kann dem Bediener damit besondere Anweisungen geben, wie er den Text zu behandeln hat. Zum Beispiel kann man einen Kommentar ans Ende des Textes stellen und dem Bediener sagen, was er als nächstes zu tun hat.

Fehlerhafte Befehle

Fehlerhafte Befehle werden als Kommentar behandelt. Der Lautsprecher ertönt und sie erscheinen auf dem Bildschirm als Kommentar, werden aber für die Textformatierung ignoriert. Dies zeigt dem Bediener, daß etwas falsch ist und zeigt die in Frage kommende Zeile. Dies geschieht, wenn beim zu formatierenden Text ein Befehlskennzeichen (gewöhnlich ein Punkt) in der ersten Spalte der Zeile steht.

Bediener-Aufforderung

Durch den Befehl *.pr* wird der Bediener aufgefordert, Informationen von Hand an einer Stelle im Text einzugeben. Der Lautsprecher ertönt, zeigt dann das Wort *enter*: gefolgt vom Text im Befehl. Wenn zum Beispiel der Bediener manuell das Datum für einen Brief eingeben muß, könnte man den Befehl *.pr Datum* an der Stelle im Text einfügen, an der das Datum erscheinen soll. Es muß, wie alle Befehle zur Formatierung, in einer eigenen Zeile stehen. Wenn dieser Befehl gefunden wird, hört der Bediener den Lautsprecher und sieht "enter: Datum" an der Konsole.

An dieser Stelle erwartet das Programm Eingaben von der Tastatur. Dieser Text wird genauso verarbeitet, als wäre er schon in der Datei vorhanden. Die Eingabe wird, wie beim Texteditor, durch eine Zeile, die nur einen Punkt enthält, abgeschlossen. Im obigen Fall wäre nur eine Zeile notwendig. Dieser würde eine Endezeile folgen. Der abschließende Punkt wird bei der Textformatierung nicht berücksichtigt.

So können Befehle zur Formatierung auch manuell eingegeben werden. Das letzte Zeichen in der aktuellen Eingabezeile kann man durch DEL oder Backspace löschen; Control-X löscht die gesamte Zeile.

Text aus anderen Quellen

Oft ist es bequem Textteile anzulegen und zu speichern, die man später in verschiedenen anderen Texten wieder verwenden kann. Dies ist möglich, wenn man den Text zuerst mit dem Texteditor erfaßt und ihn dann auf Diskette speichert. In der Datei, in der der Text erscheinen soll, gibt man an geeigneter Stelle den Befehl *.so Datei*, worauf FMT die genannte Datei an diesem Punkt in den Text kopieren wird. Dieser Befehl entspricht dem Aufforderungsbefehl, nur daß der Text hier nicht von der Tastatur sondern aus einer Datei kommt.

Der Befehl *.so* ist praktisch, wenn man Texte benötigt, die zu großen Teilen aus einzelnen Textbausteinen bestehen. Der Text in der Datei wiederum kann auch Befehle zur Formatierung oder zur manuellen Eingabe enthalten.

Zeilenabstand

Standardmäßig erzeugt FMT eine einzeilige Ausgabe. Dies kann durch den Befehl *.ls* # in einen Abstand von # Zeilen geändert werden. Ein Wert von 2 gibt doppelten Abstand, also eine Leerzeile zwischen den gedruckten Zeilen.

Gelegentlich sind zusätzliche Leerzeilen an ausgewählten Stellen im Text erwünscht. Dafür kann der Befehl *.sp* # benutzt werden. Der Befehl *.sp 11* bewirkt eine Zeilenunterbrechung gefolgt von 11 Leerzeilen.

Eine andere Möglichkeit ist, 11 Leerzeilen in den Text einzugeben. Dieser Ansatz ist jedoch nicht empfehlenswert, wenn mehr als eine Zeile benötigt wird, weil es nicht immer ganz einfach ist, vom Bildschirm auf die wirklich erzeugte Zahl der Zeilen zu schließen. Sechs Leerzeilen sehen fast genauso wie fünf aus.

Text zusammenhalten

Es gibt zwei Methoden, um zu verhindern, daß Text auf verschiedenen Seiten erscheint. Manchmal muß eine Tabelle auf einer einzigen Seite erscheinen. Um das zu garantieren, muß der Befehl *.ne* # verwendet werden, der besagt, daß für den folgenden Text # Zeilen gebraucht werden. Wenn auf der aktuellen Seite noch genug Platz ist, wird nichts unternommen. Wenn dagegen nicht genügend Platz vorhanden ist, erzwingt FMT einen Seitenvorschub und der folgende Text erscheint auf der neuen Seite.

Die zweite Methode vermeidet einzelne Zeilen am Ende einer Seite. Der Befehl für die kleinste Absatzlänge (*.mp* #) bestimmt die minimale Anzahl von Zeilen, die auf einer Seite sein müssen, bevor ein neuer Absatz gedruckt wird. Wenn nicht genug Platz vorhanden ist, beginnt eine neue Seite. Der Standardwert ist 2, der an jeder Stelle geändert werden kann.

FMT hat keine Möglichkeit, einzelne Zeilen am Beginn einer Seite zu verhindern. Man kann den Text prüfen und dann für diesen Zweck den Befehl *.ne* # einfügen.

Befehle zur Formatierung

Befehle zur Formatierung erscheinen im zu verarbeitenden Text. Jeder steht für sich auf einer einzigen Zeile. Befehle können in Groß- oder Kleinbuchstaben angegeben werden.

Die eckigen Klammern, die in der folgenden Befehlsübersicht erscheinen, sind nicht Teil der Befehle; sie zeigen nur an, daß das eingeschlossenen Feld optional ist. Nummernzeichen stehen für Zahlen zwischen 0 und

32767. Jedem numerischen Wert kann ein Plus- oder Minuszeichen vorausgehen. Die Wirkung eines Vorzeichens besteht darin, das die Zahl als Änderung einer schon bestehenden und nicht als neuer absoluter Wert interpretiert wird. Zum Beispiel bedeutet *.lm 5*, daß der linke Rand auf Spalte 5 gesetzt wird; *.lm +5* bedeutet, daß der linke Rand 5 Stellen nach rechts gebracht wird. Das Fragezeichen steht für jedes druckbare Zeichen. Die Wörter *Text* und *Datei* sind Platzhalter, die eine Textzeile oder einen Dateinamen bezeichnen.

Die Befehle zur Formatierung folgen in alphabetischer Ordnung:

`.bc ?` Pseudo-Leerzeichen einstellen
Benutzt ? für ein neues Pseudo-Leerzeichen. Vorkommen von ? werden im Text beim Zeilenfüllen und beim Blocksatz nicht als Leerzeichen behandelt, werden aber vor der Ausgabe in Leerzeichen umgewandelt.

`.bf #` Fett- oder Doppeldruck
Druckt die nächsten # Zeilen fett oder doppelt. Der Standardwert für # ist 1. *.bf 0* beendet den Fettdruck.

`.bp [#]` Neue Seite beginnen
Erzwingt Seitenvorschub. Wenn eine Zahl angegeben ist, bestimmt sie die neue Seitenzahl; sonst wird die aktuelle Seitenzahl um 1 erhöht.

`.br` Neue Zeile beginnen
Erzwingt Zeilenvorschub. Die aktuelle Zeile wird ohne Blocksatz gedruckt und eine neue Zeile beginnt.

`.cc ?` Befehlszeichen einstellen
Verwendet zukünftig ? als Zeichen für Befehle. Den folgenden Befehlen muß anstatt des üblichen Punktes ein ? vorausgehen .

`.ce [#]` Zentrieren
Erzwingt eine neue Zeile und bewirkt, daß die nächsten # Zeilen zentriert gedruckt werden. Der Standardwert für # ist 1.

`.cu [#]` Durchgehend unterstreichen
Unterstreicht die folgenden # Zeilen inklusive der Leerzeichen. Wenn # Null ist, wird das Unterstreichen beendet. Der Standardwert für # ist 1.

`.dw [#]` Doppelt breite Zeichen
Drucke die nächsten # Zeilen in doppelter Breite. Der Standardwert für # ist 1. Der Befehl *.dw 0* beendet das Drucken in doppelter Breite. Ist nur im Epson-Modus wirksam.

`.ef [Text]` Fußzeile für gerade Seiten
Text wird als Fußzeile auf jeder geraden Seite benutzt, beginnend mit der nächsten. Wenn kein *Text* vorhanden ist, ist die Fußzeile leer, es wird also eine Leerzeile gedruckt.

`.eh [Text]` Kopfzeile für gerade Seiten
Text wird als Kopfzeile auf jeder geraden Seite benutzt, beginnend mit der nächsten. Wenn kein *Text* vorhanden ist, ist die Kopfzeile leer, es wird also eine Leerzeile gedruckt.

`.fi` Zeilen füllen
Erzwingt eine neue Zeile und beginnt die nächste Zeile mit Text zu füllen. Zeilenfüllen ist der Standardmodus, also kann dieser Befehl den Modus wieder setzen, wenn er zuvor beendet wurde.

`.fo [Text]` Fußzeile
Text wird als Fußzeile auf jeder der folgenden Seite benutzt, beginnend mit der aktuellen. Wenn kein *Text* vorhanden ist, wird die Fußzeile unterdrückt, beginnend mit der aktuellen. Unterdrückte Fußzeilen erscheinen als Leerzeilen.

`.he [Text]` Kopfzeile
Text wird als Kopfzeile auf jeder der folgenden Seiten benutzt, beginnend mit der aktuellen. Wenn kein *Text* vorhanden ist, wird die Kopfzeile unterdrückt, beginnend mit der aktuellen. Unterdrückte Kopfzeilen erscheinen als Leerzeilen.

`.in [#]` Einrücken
Erzwingt eine neue Zeile und rückt dann den folgenden Text um # Stellen nach rechts ein. Wenn # ein Vorzeichen hat, wird der aktuelle Wert um den angegeben Wert unter Berücksichtigung des Vorzeichens geändert (minus nach links, plus nach rechts). Wenn der Wert negativ wird, beginnt der Druck vor dem linken Rand. Der Standardwert von # ist 0.

`.it #` Kursiv
Die nächsten # Zeilen werden kursiv gedruckt. Der Standardwert für # ist 1. Der Befehl *.it 0* beendet den Kursivdruck. Im TTY-Modus wird dieser Befehl ignoriert (wenn nicht der Schalter *-u* ein Unterstreichen bewirkt).

`.ju` Blocksatz
Erzwingt eine neue Zeile und beginnt mit dem Blocksatz in der folgenden Zeile. Da dieser Modus Standard ist, würde dieser Befehl den Modus wieder einstellen, nachdem er abgeschaltet war.

`.lm [#]`	linker Rand Erzwingt eine neue Zeile und verwendet dann # als neuen Wert für den linken Rand. Der linke Rand ist die Spalte, in der das erste Zeichen erscheint, wenn nicht eingerückt wird. Der Standardwert für # ist 11.
`.ls [#]`	Zeilenabstand Benutzt # als neuen Wert für den Zeilenabstand. Der Standardwert für # ist 1.
`.m1 [#]`	Rand 1 Benutzt # als Zahl der Zeilen in Rand 1. Der Standardwert für # ist 1. Dieser Befehl sollte *.pl* # vorausgehen.
`.m2 [#]`	Rand 2 Benutzt # als Zahl der Zeilen in Rand 2, dem Rand zwischen Rand 1 und dem eigentlichen Text. Der Standardwert für # ist 2. Dieser Befehl sollte *.pl* # vorausgehen.
`.m3 [#]`	Rand 3 Benutzt # als Zahl der Zeilen in Rand 3, dem Rand zwischen Rand 1 und dem eigentlichen Text. Der Standardwert für # ist 2. Dieser Befehl sollte *.pl* # vorausgehen.
`.m4 [#]`	Rand 4 Benutzt # als Zahl der Zeilen in Rand 4, dem Rand am Ende. Der Standardwert für # ist 9. Dieser Befehl sollte *.pl* # vorausgehen.
`.mc ?`	Feldtrennzeichen in Datendateien Verwendet *?* als Trennzeichen in den Zeilen der Datendatei. Wenn nicht anders angegeben, wird der senkrechte Strich genommen.
`.mp [#]`	Minimaler Platz für Absatz Benutzt # als die Anzahl der Zeilen eines Absatzes, die auf eine Seite passen müssen. Wenn nicht genug Platz vorhanden ist, beginnt eine neue Seite, bevor ein neuer Absatz ausgegeben wird. Der Standardwert für # ist 2.
`.ne [#]`	Zeilen zusammenhalten Erzwingt eine neue Zeile und stellt dann sicher, daß mindestens # Zeilen zusammenhängend verfügbar sind. Wenn nicht genug Platz auf der Seite ist, beginnt eine neue Seite. Der Standardwert ist Null, der Befehl wird also ignoriert.
`.nf`	Nicht füllen Erzwingt eine neue Zeile und füllt dann keine Zeilen mehr. Dies bewirkt auch, daß der Blocksatz beendet wird. Wenn Blocksatz wirksam war, wird er wieder aufgenommen, wenn die Zeilen wieder gefüllt werden.

`.nj` kein Blocksatz
Erzwingt eine neue Zeile und beendet den Blocksatz. Dieser Befehl berührt das Zeilenfüllen nicht, man erhält nur Flattersatz.

`.nu` Nicht unterstreichen
Beendet das Unterstreichen im Text, gleichgültig ob durchgehend oder nicht.

`.of [Text]` Fußzeile für ungerade Seiten
Text wird als Fußzeile auf jeder ungeraden Seite benutzt, beginnend mit der nächsten. Wenn kein *Text* vorhanden ist, ist die Fußzeile leer, also wird eine Leerzeile gedruckt.

`.oh [Text]` Kopfzeile für ungerade Seiten
Text wird als Kopfzeile auf jeder ungeraden Seite benutzt, beginnend mit der nächsten. Wenn kein *Text* vorhanden ist, ist die Kopfzeile leer, also wird eine Leerzeile gedruckt.

`.pl [#]` Seitenlänge
Benutzt # als Wert für die Seitenlänge. Der Standardwert für # ist 66. Die minimale Seitenlänge ist die Summe aus *.m1*, *.m2*, *.m3*, und *.m4* plus 1. Also müssen die Ränder vor diesem Befehl gesetzt werden.

`.po [#]` Seitenoffset
Benutzt # als neuen Wert für den Seitenoffset. Beim Druck auf ungeraden Seiten wird um diesen Wert nach rechts verschoben und auf geraden Seiten nach links. Diese Funktion kann man direkt beim Aufruf von FMT in der Befehlszeile durch *-PO#* angeben. der Standardwert für # ist 0.

`.pr [Text]` Eingabe-Aufforderung
Fordert den Bediener zur Eingabe auf. Der Lautsprecher ertönt und auf dem Bildschirm erscheint die Meldung "enter: Text" wobei das Wort *Text* für den Text steht, der dem Befehl folgt. Die Eingabe von der Tastatur wird genauso verarbeitet wie aus einer Datei. Die Eingabe wird durch eine Zeile beendet, in der nur ein Punkt steht. Diese Zeile wird bei der Textformatierung nicht berücksichtigt.

`.rm [#]` Rechter Rand
Benutzt # als neuen Wert für den rechten Rand. Der rechte Rand ist die Spalte, in der das letzte Zeichen erscheint, wenn Blocksatz wirksam ist. Der Standardwert für # ist 11.

`.rs [#]` Platz reservieren
Hält # Zeilen frei. Wenn nicht genug Zeilen auf der aktuellen Seite vorhanden sind, wird eine neue Seite begonnen und dann die angeforderte Zeilenzahl freigelassen. Höchstens eine Seite kann freigehalten werden.

`.so Datei` Quelldatei für Text
Benutzt die genannte Datei an diesem Punkt als Quelldatei. Wenn die Datei komplett gelesen wurde, mit der nächsten Zeile fortgefahren.

`.sq [#]` Zusammendrücken
Erzwingt eine neue Zeile und rückt dann von beiden Ränder um # Stellen nach innen ein. Der Standardwert für # ist Null. Ab dem Befehl *.sq* wird wieder normal gedruckt. Der Befehl *.sq +#* addiert zum schon wirksamen Wert und *.sq -#* subtrahiert davon.

`.sp [#]` Leerzeilen
Erzwingt eine neue Zeile und übergeht dann # Zeilen. Der Standardwert für # ist 1.

`.ti [#]` Zeitweise einrücken
Erzwingt eine neue Zeile und nimmt dann als Einrückwert #, aber nur für die nächste Zeile. Folgende Zeilen verwenden wieder dem laufenden Wert. Durch ein Plus oder Minus vor # wird dieser Wert zum laufenden addiert oder subtrahiert, Der Standardwert für # ist 0, der Befehl wird also ignoriert.

`.ul [#]` Nicht durchgehend unterstreichen
Unterstreicht die folgenden # Zeilen ohne die Leerzeichen zwischen den Wörtern. Wenn # Null ist, wird das Unterstreichen beendet. Der Standardwert für # ist 1.

`.. [Text]` Kommentar
Zwei aufeinanderfolgende Befehlszeichen kennzeichnen eine Kommentarzeile, die von FMT nicht beachtet wird. Man kann zeitweise einen Befehl unwirksam machen, indem man ein weiteres Befehlszeichen davor setzt.

Meldungen

MELDUNG ERKLÄRUNG

`ready printer...`
Der Bediener soll sicherstellen, daß Papier eingespannt ist und der Drucker auf Online steht und dann RETURN oder ENTER drücken.

`set page #`
Der Bediener soll das nächste Blatt einspannen und dann auf RETURN oder ENTER drücken (Control-N geht direkt zur nächsten Seite).

`copy # ready printer`
Bereite den Drucker für Kopie # vor. (Control-N geht direkt zur nächsten Seite).

`page #`

Sagt dem Bediener, welche Seite gerade gedruckt wird, so daß ein Neustart auf der letzten gedruckten Seite möglich ist.

`enter: <Beschreibung>`

Der Bediener soll Text eingeben und die Eingabe dann mit einer Zeile abschließen, die nur einen Punkt enthält.

`note: <Zeile>`

Die angezeigte Zeile ist entweder ein Kommentar, ein falscher Formatierbefehl oder eine Textzeile, die unbeabsichtigt mit einem Punkt beginnt (oder einem anderen Befehlszeichen).

`error: <Zeile>`

Die angezeigte Zeile enthält einen Formatierbefehl mit einem numerischen Argument, der nicht ausgewertet werden kann.

LST (List)

LST [Datei] [-C#] [-PL#] [-PW#] [-NB] [-NN] [-NP]

Beschreibung

LST kopiert eine Eingabedatei zur Standardausgabe. Wenn eine Datei in der Befehlszeile angegeben ist, wird sie als Eingabe genommen, sonst wird die Standardeingabe angenommen. Wahlweise werden die Zeilen numeriert, erscheint die Ausgabe in mehreren Spalten pro Seite und/oder es wird eine Pause zwischen den Seiten gemacht. LST hat den Hauptzweck, Textdateien am Bildschirm auszugeben, man kann es jedoch auch in Verbindung mit FMT oder PRT benutzen, um Listings auf dem Drucker zu erhalten, deren Zeilen numeriert sind und/oder in mehreren Spalten aufgeteilt sind.

Der Schalter -C # sagt LST, wieviele Spalten pro Seite in der Ausgabe erscheinen sollen. Der Standardwert ist eins. LST teilt die Seite durch #, um die Breite der einzelnen Spalten zu bestimmen. Wenn eine Zeile zu lang ist um in die Spalte zu passen, erscheint sie in der nächsten in der gleichen Spalte. Normalerweise werden alle Zeilen kürzer als die Spaltenbreite sein; wenn jedoch eine Zeile genauso lang ist wie die Spalte breit ist, erhält man einen Spaltenüberlauf, bevor das Ende der Zeile entdeckt wurde. Als Ergebnis erhält man eine leere Zeile in der Spalte. Man kann diese Leerzeilen durch den Schalter -NB entfernen.

Der Schalter -PL # gibt LST die Seitenlänge in Zeilen an. Dies ist nicht die physikalische Länge einer Seite, sondern die Zahl der Zeilen, die LST auf die Seite druckt. Die Standard-Seitenlänge hängt davon ab, ob die Ausgabe auf die Diskette umgeleitet wurde. Wenn ja, wird die Länge aus PTRHIGH-PTRSKIP-PTRHDR (identisch mit den von PRT benutzten) berechnet; sonst werden CRTHIGH-1 Zeilen (eine weniger als die Länge des Bildschirms) angenommen. Diese Symbole sind in TOOLS.H definiert und können an die Erfordernisse angepaßt werden (das Programm muß dann neu kompiliert werden).

Der Schalter -PW# gibt LST die Breite einer Seite in Zeichen an. Auch der Standardwert für die Breite einer Seite hängt davon ab, ob die Ausgabe zur Diskette umgeleitet wurde oder nicht. Wenn ja, wird PTRWIDE-1 angenommen; sonst CRTWIDE-1. Diese Symbole sind in TOOLS.H definiert und können an die Erfordernisse angepaßt werden.

Sofern man LST nichts anderes sagt, werden die Zeilen numeriert. Mit -NN (no numbers) schaltet man die Numerierung aus. Zeilennummern erscheinen rechtsbündig mit Leerstellen aufgefüllt in einem vier Zeichen breiten Feld am Anfang jeder Spalte. Wenn die Zeilennummerierung unterdrückt wird, kann die gesamte Spaltenbreite für Text genutzt werden.

Wenn man nichts anderes angibt, macht LST zwischen den Seiten eine Pause und wartet auf Antwort, bevor es weitermacht. Dadurch hat man genug Zeit den Bildschirminhalt zu studieren, bevor er wegscrollt. Bei der Ausgabe auf den Drucker kann man einzelne Blätter verwenden. Wenn man keine Pause haben möchte kann man den Schalter -NP angeben (no pause). In diesem Fall fährt LIST automatisch fort und läßt keine Leerzeilen zwischen den Seiten. Die Ausgabe erscheint immer noch in (möglicherweise) mehreren Spalten, aber ohne Leerzeilen zwischen den Seiten. Wenn die Ausgabe auf Diskette erfolgt, gibt es niemals Pausen zwischen den Seiten. Es ist in diesem Fall nicht notwendig, den -NP Schalter anzugeben.

Die letzte Zeile einer Seite befindet sich am Ende der am weitesten rechts stehenden Spalte und die erste Zeile der nächsten Seite befindet sich am Anfang der am weitesten links stehenden Spalte.

Wenn man einen Seitenvorschub vor dem Drucken braucht, kann man die Ausgabe zur Diskette schicken und diese Datei dann als Eingabe für PRT benutzen. Die Standard-Seitenlänge stimmt überein, so daß der Vorschub immer genau an der richtigen Stelle erfolgt.

LST ist besonders nützlich, wenn man sich kurze Textzeilen anschauen möchte, Zeilen, die nur aus einem Wort bestehen, wie man sie etwa in Wörterbüchern findet. Es spart eine Menge Papier und macht auch das Ansehen am Bildschirm viel leichter.

Beispiele

BEFEHL KOMMENTAR

`LST <ABC -C3`

Gibt auf dem Bildschirm den Inhalt der Datei ABC in drei Spalten mit Zeilennummerierung und Pause zwischen den Seiten aus.

`LST <ABC >DEF -C8 -PW132 -NP`

Kopiert die Datei ABC nach DEF, numeriert die Zeilen und bringt sie in ein acht-spaltiges Format auf eine Seite, die 132 Zeichen breit ist. PRT kann dann benutzt werden, um die Datei DEF mit Seitenvorschub und Kopfzeilen zu auszudrucken.

Meldungen

MELDUNG ERKLÄRUNG

`waiting...`

LST wartet auf die Antwort des Bedieners mit Return oder Enter, bevor die nächste Seite angezeigt wird.

MRG (Merge)

`MRG Datei [Datei] [-1|-2|-3|-F]`

Beschreibung

MRG nimmt zwei sortierte Textdateien, führt sie zusammen und gibt das Ergebnis aus. Die Eingabedateien müssen vollständig aufsteigend sortiert sein. Verglichen wird lexikographisch, also haben Klein- und Großbuchstaben den gleichen Rang und Sonderzeichen haben einen niedrigere Wert als Buchstaben.

Die Ausgabe kann aus beiden Dateien bestehen oder aus Zeilen, die nur in der ersten oder nur in der zweiten vorkommen.

Die erste genannte Datei in der Befehlszeile wird als erste Datei bezeichnet. Die nächste angegebene Datei heißt die zweite Datei. Wenn eine zweite Datei nicht angegeben wird, wird stattdessen die Standardeingabe angenommen. Natürlich kann die Standardeingabe zur Diskette umgeleitet werden; in diesem Fall kann die Umlenkungsanweisung (zweite Datei) irgendwo in der Befehlszeile stehen, sogar vor der ersten Datei.

Wenn nicht anders angegeben, gibt MRG den gesamten Inhalt beider zusammengeführter Dateien aus. Übereinstimmende Zeilen in den beiden Zeilen erscheinen jedoch nur einmal in der Ausgabe.

Wenn man den Schalter -1 angibt, werden nur die Zeilen aus der ersten Datei ausgegeben, die nicht mit der zweiten Datei übereinstimmen; Zeilen aus der zweiten Datei erscheinen überhaupt nicht.

Wenn man den Schalter -2 angibt, werden nur die Zeilen aus der zweiten Datei ausgegeben, die nicht mit der ersten Datei übereinstimmen; Zeilen aus der ersten Datei erscheinen überhaupt nicht.

Wenn man den Schalter -3 angibt, werden die Zeilen aus beiden Dateien ausgegeben, die in beiden identisch sind. Wenn diese identischen Zeilen mehrfach vorhanden sind, werden sie sooft aus derjenigen Datei ausgegeben, in der sie weniger häufig stehen.

Der Schalter -F bewirkt, daß alle Zeilen formatiert ausgegeben werden. Dadurch erscheinen alle Zeilen, die nur in der ersten Datei vorkommen, am äußerst linken Rand; Zeilen, die nur in der zweiten Datei vorkommen, werden zwei Stellen eingerückt; Zeilen die in beiden Dateien übereinstimmen, werden um weitere zwei Stellen eingerückt. Um die Art der Zeilen erkennen zu können, werden den drei Spalten die Zahlen 1), 2) oder 3) vorangestellt, da es möglich sein kann, daß es nur eine Zeilenart gibt.

Beispiele

BEFEHL	KOMMENTAR
`MRG ABC DEF -1`	Führt die Dateien ABC und DEF zusammen, wobei auf dem Bildschirm nur die Zeilen von ABC gezeigt werden, die nicht in DEF vorhanden sind.
`MRG ABC DEF -F >LST:`	Führt die Dateien ABC und DEF zusammen und gibt ein formatiertes Listing auf dem logischen Gerät LST: aus.
`MRG ABC DEF >GHI`	Führt die Dateien ABC und DEF zusammen und speichert die kombinierte Ausgabe beider Dateien in der Datei GHI.

Meldungen: keine

PRT (Print)

```
PRT [Datei]... [.[?]] [-NN] [-NH|-NS] [-LM#] [-BP#]
               [-EP#] [-P]  [-NR]
```

Beschreibung

PRT kopiert eine oder mehrere Textdateien zur Standardausgabe. Wenn die Standardausgabe nicht vom Bildschirm weggelenkt wurde, wird sie geschlossen und auf LST: wieder geöffnet. Daher ist LST: die Standardausgabe für PRT. Normalerweise numeriert PRT die Zeilen durch, unterteilt die Zeilen in Seiten zu 56 Zeilen und stellt eine Kopfzeile über jede Seite mit dem Dateinamen und der Seitennummer. Die Eingabe erfolgt über die Standardeingabe, die wie immer umgeleitet werden kann. Wenn eine Liste von Dateinamen in der Befehlszeile angegeben wurde, werden sie auch in dieser Reihenfolge gedruckt, jeweils mit einem Seitenvorschub zwischen den Dateien. In diesem Fall wird die Standardeingabe nicht benutzt.

Wenn sie vorhanden sind, verarbeitet PRT auch Include-Dateien. Die Anweisung in C heißt "*#include* Datei", für die Textformatierung heißt sie "*.so* Datei". Die Datei erscheint in der Ausgabe sofort hinter den genannten Anweisungen. Die Zeilennummerierung beginnt für jede dieser Dateien wieder bei 1, geht jedoch für die erste Datei ununterbrochen weiter. Die beiden genannten Anweisungen können beliebig tief geschachtelt werden, begrenzt nur durch die Speicherkapazität.

Die Parameter, die man bei PRT angeben kann sind:

`[Datei]...`
Druckt die genannten Dateien anstatt die Standardeingabe

`.[?]`	Druckt alle eingeschlossenen Dateien mit einer Erweiterung von ?, wobei ? aus einem bis drei Zeichen besteht. Wenn ? leer ist, werden alle eingeschlossenen Dateien gedruckt.
-NN	Keine Zeilennummerierung
-NH	Keine Kopfzeilen
-NS	Keine Seiten übergehen (und keine Kopfzeilen)
-LM#	Linker Rand mit # Stellen. Der Standardwert ist 0 Stellen.
-BP#	Druckbeginn auf Seite #
-EP#	Druckende auf Seite #
-P	Pause nach jeder Seite. Standardmäßig wird fortlaufend gedruckt.
-NR	Keine "ready printer..." Aufforderung

Beispiele

BEFEHL	KOMMENTAR
`PRT ABC DEF >PUN:`	Druckt die Dateien ABC und DEF auf dem logischem Gerät PUN: mit einer Kopfzeile auf jeder Seite und einem Seitenvorschub zwischen den Dateien.
`PRT SORT.C .C`	Druckt SORT.C zusammen mit allen C-Dateien, die darin eingeschlossen sind. Die Ausgabe geht nach LST:.

Meldungen

MELDUNG	ERKLÄRUNG
`ready printer...`	PRINT wartet darauf, daß Papier eingespannt wird und auf Online geschaltet wird, damit Daten vom Computer geschickt werden können.
`page #`	Zeigt an, welche Seite gerade gedruckt wird, so daß ein Neuanfang möglich ist.

SRT (Sort)

```
SRT [-C#|-F#?] [-D] [-U] [-Tx] [-Q]
```

Beschreibung

SRT liest die Standardeingabe, sortiert sie Zeile für Zeile und schreibt die sortierten Daten zur Standardausgabe. Verarbeitet werden nur Standard-Textdateien (ASCII-Format) keine binären Dateien. Die Dateien können beliebig groß sein.

Wenn kein Sortierschlüssel angegeben wurde, wird die ganze Zeile als Schlüssel genommen. Das heißt, der Vergleich beginnt mit dem ersten Zeichen der beiden zu vergleichenden Zeilen von links nach rechts und verglichen werden die entsprechenden Zeichen solange bis ein Unterschied festgestellt wird. Die Zeile, deren nicht übereinstimmendes Zeichen lexikalisch tiefer steht, wird bei aufsteigender Sortierung zuerst ausgegeben, bei absteigender Sortierung zuletzt.

Zwischen Groß- und Kleinbuchstaben wird kein Unterschied gemacht. Sonderzeichen haben die gleiche relative Position zueinander, wie in der ASCII-Tabelle, gehen jedoch allen Buchstaben voraus. Das ASCII-Zeichen DEL (Wert 127 dezimal) hat den höchsten Wert.

Wenn eine Zeile länger als die andere ist und am Ende der kürzeren Zeile kein Unterschied gefunden wurde, wird angenommem, daß die kürzere "kleiner" als die längere ist.

Standardmäßig wird aufsteigend sortiert. Wenn jedoch der Schalter -D in der Befehlszeile erscheint, wird die Reihenfolge umgekehrt.

Wenn nur ein Teil zum Vergleich herangezogen wird, kann einer von zwei Schaltern angegeben werden, um anzuzeigen, wo der Sortierschlüssel gefunden werden kann. Der Schalter -C# informiert SRT, daß der Schlüssel in Spalte # beginnt und bis zum Zeilenende reicht. Wenn # größer ist als die Zeichen in der Zeile, wird der Sortierschlüssel als Null angenommen und solche Zeilen werden bei aufsteigender Sortierung an den Anfang sortiert.

Der Schalter -F#? sagt SRT, daß das Feld # in einer Zeile als Sortierschlüssel genommen werden kann. Ein Feld kann in unterschiedlichen Zeilen unterschiedlich lang sein und kann auch in verschiedenen Spalten beginnen. Felder werden durch ein besonderes Zeichen definiert. Dieses Zeichen steht im Schalter hinter der Feldnummer. Das erste Feld ist alles in einer Zeile bis zum, aber nicht einschließlich, ersten definierten besonderen Zeichen. Das zweite besteht aus dem Inhalt zwischen den beiden definierten Zeichen. Das Zeilenende ist zugleich auch das Ende für das letzte Feld.

Wenn zwei Zeilen verglichen werden, werden die Schlüsselfelder im allgemeinen an verschiedenen Stellen stehen und unterschiedlich lang sein. Die Vergleichsmethode entspricht der von ganzen Zeilen. Wenn ein Feld kürzer als das andere ist und die beiden Felder bis zum definierten Endezeichen des kürzeren Feldes übereinstimmen, wird das kürzere Feld niedriger in der Sortierreihenfolge eingestuft. Wenn die definierten Zeichen nebeneinander stehen, begrenzen sie ein leeres Feld, das heißt das Feld mit einer Länge von Null zählt als ein Feld. Wenn es als Schlüssel verwendet wird, erscheint es tiefer, als alle Felder mit einer Länge ungleich Null.

Wenn kein Zeichen für ? angegeben ist, werden alle Leerstellen genommen. Diese Zeichen sind alle Sonderzeichen, die zwischen Feldern mit druckbaren Zeichen stehen. In einer Textdatei werden durch diese Zeichen Wörter getrennt. Der Schalter -F5 sagt SRT demnach, daß jedes fünfte Wort in einer Zeile als Schlüssel genommen werden soll.

Da die Ausgabe von SRT identische Zeilen zusammenführt (wenn kein Sortierschlüssel angegeben ist), ist SRT auch der natürliche Ort, um doppelte Zeilen auszusondern. Durch den Schalter -U kann man genau dies erreichen. Wenn dieser Schalter angegeben ist, erscheinen nur einmalige Zeilen in der Ausgabe. Der Vergleich auf Einmaligkeit nimmt immer die gesamte Zeile, auch wenn ein Sortierschlüssel angegeben wurde.

Da die zu sortierenden Dateien größer als der verfügbare Speicher sein können, ist es oft nötig, daß SRT Zwischendateien auf der Diskette anlegt, die Teile der schon sortierten Eingabedatei enthalten, während der Rest noch weiter sortiert wird. Es werden automatisch so viele Zwischendateien angelegt, wie für SRT erforderlich sind. Am Ende der Eingabe werden die Zwischendateien für die Ausgabe zusammengeführt. Anschließend werden die Zwischendateien gelöscht. Die Namen dieser Dateien sind SORT01. $$$, SORT02.$$$ und so weiter. Bei Unterbrechung eines Sortierlaufs können diese Dateien auf der Diskette stehen bleiben. Man sollte sie dann löschen.

Wenn nicht anders angegeben, werden die Zwischendateien auf dem Standardlaufwerk angelegt. Der Schalter -Tx (x gibt ein Laufwerk von A bis G an) legt ein besonderes Laufwerk für die Zwischendateien fest.

SRT benutzt normalerweise den Shell-Sortieralgorithmus für die Sortierung der Zeilen im Speicher. Ein zweiter Algorithmus, Quick-Sort, ist auch verfügbar, der durch den Schalter -Q aufgerufen wird. Dieser Algorithmus ist schneller, hat aber auch Nachteile. Wenn die Eingabe schon sortiert ist, wird er sehr viel langsamer als der Shell-Sort und benutzt auch sehr viel mehr Speicher, so daß es zu einem Speicherzuordnungsfehler kommen kann. Wenn dies geschieht, wird ein M angezeigt und der Lauf beendet.

Da die meiste Zeit in einem Sortierlauf bei der Ein- und Ausgabe verbraucht wird, ist die größere Schnelligkeit des Quick-Sorts kein großer Vorteil; er ist jedoch für Systeme mit größerem Speicher und/oder schnellen Festplatten beibehalten worden.

Beispiele

BEFEHL KOMMENTAR

SRT <ABC >DEF -F5|

Sortiert die Datei ABC nach dem 5ten Feld, getrennt durch das Zeichen | und schreibt die Ausgabe in die Datei DEF.

SRT <ABC -U

Sortiert die Datei ABC nach ganzen Zeilen und zeigt nur einmalige Zeilen an der Konsole an.

Meldungen

MELDUNG ERKLÄRUNG

file too large

Die Eingabedatei ist zu groß, da mehr als 99 Zwischendateien erforderlich sind.

TRN (Trans)

TRN [~]von [nach]

Beschreibung

TRN kopiert die Standardeingabe zur Standardausgabe und übersetzt dabei ausgewählte Zeichen in neue Werte. Der Parameter *von* ist eine Liste der zu ändernden Zeichen. Zwischen den Zeichen in der Liste dürfen keine Leerzeichen stehen. In *nach* stehen die neuen Zeichen, die den aus *von* zugewiesen werden sollen. Das erste *von*-Zeichen entspricht dem ersten *nach*-Zeichen, das zweite *von*-Zeichen, dem zweiten *nach*-Zeichen und so fort.

Demnach kopiert der Befehl "TRN <DATEI1 >DATEI2 abc ABC" die Datei DATEI1 nach DATEI2 und wandelt die Kleinbuchstaben *abc* in Großbuchstaben *ABC* um.

Wichtig: Man muß SUBMIT und den CCP patchen, damit Kleinbuchstaben akzeptiert werden können. Genaueres ist im Anhang A zu finden.

Damit es einfacher ist, das ganze Alphabet oder Teile davon anzugeben, braucht nur der erste und der letzte Buchstaben, durch Bindestrich verbunden, eingegeben werden. Durch diese Abkürzung werden alle dazwischen liegendenen Buchstaben spezifiziert. Also kopiert der Befehl "TRN <DATEI1 >DATEI2 a-z A-Z" die Datei DATEI1 nach DATEI2 und wandelt alle Kleinbuchstaben in Großbuchstaben um. Der Befehl "TRN a-zA-Z A-Za-z <DATEI1 >DATEI2" wandelt beim Kopieren von DATEI1 nach DATEI2 alle Kleinbuchstaben in Großbuchstaben um und umgekehrt. Der Befehl "TRN <DATEI1 a-c d-f >DATEI2" wandelt während des Kopierens von DATEI1 nach DATEI2, a nach d, b nach e und c nach f um.

Escapesequenzen können in den Listen *von* und *nach* benutzt werden. Da beide dieser Listen durch ein Leerzeichen oder Tab-Zeichen beendet werden, müssen diese Werte in der Liste mit :s und :t angegeben werden. Ein Doppelpunkt muß doppelt eingegeben werden.

Wenn die *nach*-Liste länger ist als die *von*-Liste, werden die restlichen Zeichen nicht weiter beachtet.

Wenn auf der anderen Seite die *nach*-Liste kürzer als die *von*-Liste ist, operiert TRN zusammenschiebend. Die *nach*-Liste wird automatisch auf die Länge der *von*-Liste erweitert, indem das letzte Zeichen der *nach*-Liste wiederholt wird. Anschließend schiebt TRN alle vorkommenden Zeichen zu einem einzigen Zeichen zusammen. Wenn man zum Beispiel jedes Wort in einem Text auf je einer Zeile haben möchte, genügt dafür der Befehl "TRN <DOC1 >DOC2 :s:t :n". Leerzeichen und Tabs würden beide in Zei-

chen für Neue-Zeile umgewandelt, dann würden diese aufeinanderfolgenden Zeichen zu einem Zeichen zusammengeschoben. Damit verhindert man in der Ausgabe Leerzeilen.

Um es zu wiederholen, TRN schiebt das letzte Zeichen in der *nach*-Liste zusammen, wenn die *nach*-Liste kürzer als die *von*-Liste ist. Da dies gilt, wenn die *nach*-Liste mehr als ein Zeichen enthält, ist es möglich, einige Zeichen umzuwandeln, während andere zusammengeschoben werden. In dem Sonderfall, wenn die *nach*-Liste vollständig entfällt, werden übereinstimmende Zeichen aus der *von*-Liste in der Ausgabe gelöscht.

Manchmal möchte man alle bis auf ganz bestimmte Zeichen löschen oder zusammenschieben. Für diesen Zweck muß die Tilde (~), die *nicht* bedeutet, der *von*-Liste vorangestellt werden.

Wenn dem Programmnamen in der Befehlszeile ein Bindestrich allein folgt, wird dieser als Schalter ohne Bedeutung interpretiert, das heißt der Bedienungshinweis wird angezeigt. Möchte man also Bindestriche aus der Datei entfernen, muß der Binsdestrich in der Befehlszeile als Escape-Sequenz erscheinen.

Beispiele

BEFEHL KOMMENTAR

TRN <ABC >DEF ~[a-z][A-Z] :n

Kopiert die Datei ABC nach DEF, schiebt alle Nichtbuchstaben zu Neue-Zeile-Zeichen zusammen (also wird alles außer Wörtern entfernt, so daß jedes Wort in einer Zeile steht).

TRN <ABC :[:]{}()

Zeigt auf dem Bildschirm nur die eckigen, runden und geschweiften Klammern aus der Datei ABC an.

Meldungen

MELDUNG ERKLÄRUNG

from-list too large

Die intern erweiterte *von*-Liste ist zu groß, um in den reservierten Speicherplatz zu passen.

to-list too large

Die intern erweiterte *nach*-Liste ist zu groß, um in den reservierten Speicherplatz zu passen.

6 Kompilierung der Quellprogramme

Es ist natürlich aufgrund der Vielzahl der Systeme, auf denen das Small-C-Entwicklungssystem läuft, unmöglich, für alle Gegebenheiten eine genaue Anleitung zur Art und Weise der Kompilierung und vor allem der Diskettenaufteilung zu geben. Deshalb beschränkt sich diese Beschreibung auf ein sehr einfaches System mit 64 KByte Speicher, einem Diskettenlaufwerk (ca. 160 KByte) und dem Betriebssystem CP/M Plus.

Zunächst wird anhand von zwei einfachen Beispielen aus den Small-Tools gezeigt, wie man ein C-Programm kompiliert und wie man bei der genannten Konfiguration die Disketten so aufteilt, daß auch größere Programme kompiliert werden können.

Diskettenaufteilung

Damit alle Quellprogramme des Small-C-Entwicklungssystems kompiliert werden könnne, müssen bei der angenommenen Konfiguration die benötigten Programme auf drei Disketten verteilt werden. Zusätzlich wird auf jeder Diskette das Kopierprogramm PIP benötigt oder unter CP/M 2.2 ein anderes Kopierprogramm, das es erlaubt, Dateien mit einem Laufwerk von einer Diskette zu einer anderen zu kopieren (zum Beispiel FILECOPY bei einigen Schneider-Computern). Es folgt eine Aufstellung der Disketteninhalte und von welcher Diskette diese Programme kopiert werden müssen.

Diskette 1: C-Compiler

```
PIP.COM   von der Betriebssystem-Diskette
CC.COM    von der Diskette SC1
STDIO.H   von der Diskette SC1
```

Diskette 2: Small-Mac-Assembler

```
PIP.COM   von der Betriebssystem-Diskette
MAC.COM   von der Diskette SM1
```

Diskette 3: Linker

```
PIP.COM   von der Betriebssystem-Diskette
LNK.COM   von der Diskette SM1
C.LIB     von der Diskette SM1
C.NDX     von der Diskette SM1
```

Ein einfaches Beispiel

Im ersten Beispiel soll das Small-Tools-Programm CPT (Crypt) kompiliert werden, denn es ist das kleinste lauffähige Programm im Small-C-Entwick-

lungssystem und benötigt darüber hinaus keine besonderen Include-Dateien. Kopieren Sie also nun noch die Datei CPT.C von der Diskette ST auf die Diskette 1 (C-Compiler). Das Programm CPT sieht folgendermaßen aus:

```
/*
** cpt.c -- encrypt or decrypt ASCII or binary files
**
** Copyright 1982 J. E. Hendrix.  All rights reserved.
*/
#include <stdio.h>
#define NOCCARGC
#define MAXKEY 81
#define CTLZ   26
main(argc, argv) int argc, *argv; {
  char c, key[MAXKEY];
  int i, keylen;
  auxbuf(stdin, 4096);
  keylen=getarg(1, key, MAXKEY, argc, argv);
  if((keylen==EOF)|(key[0]=='-')) {
    fputs("usage: CPT key\n", stderr);
    abort(7);
    }
  i=1;
  while(read(stdin, &c, 1) > 0) {
    poll(YES);
    if(isatty(stdin) && (c==CTLZ)) break;
    c=c^key[i-1];
    if(write(stdout, &c, 1) !=1 ) {
      fputs("output error\n", stderr);
      abort(7);
      }
    i=(i%keylen)+1;
    }
  }
```

Geben Sie nun unter CP/M folgenden Befehl ein:

```
A>CC -M CPT
```

Dadurch wird der Small-C-Compiler aufgerufen. Der Parameter -M bedeutet, daß der Compiler während der Bearbeitung die erste Zeile jeder Funktion auf dem Bildschirm ausgegebn soll (Monitor). Dadurch weiß man dann immer, wie weit der Compiler mit seiner Arbeit fortgeschritten ist und kann zusätzlich eventuelle Fehler besser lokalisieren. CPT ist der Name des zu kompilierenden Programms. Die Dateinamenserweiterung C braucht nicht angegeben zu werden. Der Compiler zeigt nun beim Kompilieren folgendes an:

```
Small-C Compiler, Version X.X (Rev. XX)
Copyright 1982, 1983 J. E. Hendrix

main(argc, argv) int argc, *argv; {

A>
```

Nach einigen Sekunden meldet sich das Betriebssystem und das Programm ist in 8080-Assemblercode übersetzt. Der Assemblercode steht auf der Diskette nun in der Datei CPT.MAC. Bitte sehen Sie sich diese Datei einmal mit dem Betriebssystembefehl TYPE an. Das sieht dann etwa so aus:

```
CC1:
main::
LXI H,-86
DAD SP
SPHL
LXI H,0
PUSH H
LXI H,4096
PUSH H
CALL auxbuf
POP B
POP B
LXI H,0
DAD SP
PUSH H
LXI H,1
PUSH H
LXI H,8
DAD SP
PUSH H
LXI H,81
PUSH H
LXI H,98
DAD SP
CALL CCGINT##
PUSH H
LXI H,98
DAD SP
CALL CCGINT##
PUSH H
CALL getarg
XCHG;;
.
.
.
.
.
DAD SP
CALL CCGINT##
XCHG;;
POP B
POP H
PUSH H
PUSH B
CALL CCDIV##
XCHG;;
LXI D,1
DAD D
POP D
CALL CCPINT##
JMP CC4
CC5:
LXI H,86
DAD SP
SPHL
RET
CC2:DB 117,115,97,103,101,58,32,67,80,84
DB 32,107,101,121,10,0,111,117,116,112
DB 117,116,32,101,114,114,111,114,10,0
EXT read
EXT isatty
EXT poll
EXT auxbuf
EXT abort
EXT fputs
EXT write
EXT getarg
EXT Ulink
END
```

Kopieren Sie diese Datei nun mit dem Programm PIP auf die Diskette 2 (Small-Mac-Assembler). Geben Sie hier nun folgenden Befehl ein:

```
A>MAC -L CPT
```

Damit wird der Small-Mac-Makroassembler aufgerufen. Der Parameter -L bedeutet, daß er ein Listing ausgeben soll. CPT ist der Name der vom Compiler erzeugten Assemblercode-Datei. Auch hier braucht die Namenserweiterung .MAC nicht angegeben werden. Der Assembler meldet sich nun mit seiner Startmeldung:

```
Small-Mac Assembler, Version X.X (Rev. XX)
Copyright 1985 J. E. Hendrix
```

Dann wird das Programm so ausgegeben, wie es auf den folgenden Seiten zu sehen ist. In der ersten Zeile auf jeder Seite wird der Dateiname und die aktuelle Seitennummer ausgegeben. Für jede Assemblerzeile werden folgende Informationen ausgegeben:

- Zeilennummer (line),
- Adresse des Assemblercodes (loc),
- der erzeugte Objektcode (falls vorhanden),
- der Quellcode selbst (source).

```
file: CPT.MAC    page:    1

line  loc ----object----  source

   1    0                 CC1:
   2    0                 main::
   3    0 21 FFAA         LXI H,-86
   4    3 39              DAD SP
   5    4 F9              SPHL
   6    5 21 0000         LXI H,0
   7    8 E5              PUSH H
   8    9 21 1000         LXI H,4096
   9    C E5              PUSH H
  10    D CD 0000         CALL auxbuf
  11   10 C1              POP B
  12   11 C1              POP B
  13   12 21 0000         LXI H,0
  14   15 39              DAD SP
  15   16 E5              PUSH H
  16   17 21 0001         LXI H,1
  17   1A E5              PUSH H
  18   1B 21 0008         LXI H,8
  19   1E 39              DAD SP
  20   1F E5              PUSH H
  21   20 21 0051         LXI H,81
  22   23 E5              PUSH H
  23   24 21 0062         LXI H,98
  24   27 39              DAD SP
  25   28 CD 0000         CALL CCGINT##
  26   2B E5              PUSH H
  27   2C 21 0062         LXI H,98
  28   2F 39              DAD SP
  29   30 CD 0029'        CALL CCGINT##
  30   33 E5              PUSH H
  31   34 CD 0000         CALL getarg
  32   37 EB              XCHG;;
  33   38 21 000A         LXI H,10
  34   3B 39              DAD SP
  35   3C F9              SPHL
  36   3D EB              XCHG;;
  37   3E D1              POP D
  38   3F CD 0000         CALL CCPINT##
  39   42 D1              POP D
  40   43 D5              PUSH D
  41   44 21 FFFF         LXI H,-1
  42   47 CD 0000         CALL CCEQ##
  43   4A E5              PUSH H
  44   4B 21 0006         LXI H,6
  45   4E 39              DAD SP
  46   4F CD 0000         CALL CCGCHAR##
  47   52 EB              XCHG;;
  48   53 21 002D         LXI H,45
  49   56 CD 0048'        CALL CCEQ##
  50   59 D1              POP D
  51   5A CD 0000         CALL CCOR##
  52   5D 7C              MOV A,H
  53   5E B5              ORA L
  54   5F CA 0077'        JZ CC3
```

```
file: CPT.MAC    page:   2

line  loc ----object----  source

      62 21 015D'         LXI H,CC2+0
      65 E5               PUSH H
      66 21 0002          LXI H,2
      69 E5               PUSH H
      6A CD 0000          CALL fputs
      6D C1               POP B
      6E C1               POP B
      6F 21 0007          LXI H,7
      72 E5               PUSH H
      73 CD 0000          CALL abort
      76 C1               POP B
      77                  CC3:
      77 21 0002          LXI H,2
      7A 39               DAD SP
      7B EB               XCHG;;
      7C 21 0001          LXI H,1
      7F CD 0040'         CALL CCPINT##
      82                  CC4:
      82 21 0000          LXI H,0
      85 E5               PUSH H
      86 21 0057          LXI H,87
      89 39               DAD SP
      8A E5               PUSH H
      8B 21 0001          LXI H,1
      8E E5               PUSH H
      8F CD 0000          CALL read
      92 C1               POP B
      93 C1               POP B
      94 C1               POP B
      95 AF               XRA A
      96 B4               ORA H
      97 FA 0157'         JM CC5
      9A B5               ORA L
      9B CA 0157'         JZ CC5
      9E 21 0001          LXI H,1
      A1 E5               PUSH H
      A2 CD 0000          CALL poll
      A5 C1               POP B
      A6 21 0000          LXI H,0
      A9 E5               PUSH H
      AA CD 0000          CALL isatty
      AD C1               POP B
      AE 7C               MOV A,H
      AF B5               ORA L
      B0 CA 00CC'         JZ CC7
      B3 21 0055          LXI H,85
      B6 39               DAD SP
      B7 CD 0050'         CALL CCGCHAR##
      BA EB               XCHG;;
      BB 21 001A          LXI H,26
      BE CD 0057'         CALL CCEQ##
      C1 7C               MOV A,H
      C2 B5               ORA L
      C3 CA 00CC'         JZ CC7
```

```
file: CPT.MAC    page:   3

line   loc ----object----   source

 109    C6 21 0001           LXI H,1
 110    C9 C3 00CF'          JMP CC8
 111    CC                   CC7:
 112    CC 21 0000           LXI H,0
 113    CF                   CC8:
 114    CF 7C                MOV A,H
 115    D0 B5                ORA L
 116    D1 CA 00D7'          JZ CC6
 117    D4 C3 0157'          JMP CC5
 118    D7                   CC6:
 119    D7 21 0055           LXI H,85
 120    DA 39                DAD SP
 121    DB E5                PUSH H
 122    DC 21 0057           LXI H,87
 123    DF 39                DAD SP
 124    E0 CD 00B8'          CALL CCGCHAR##
 125    E3 E5                PUSH H
 126    E4 21 0008           LXI H,8
 127    E7 39                DAD SP
 128    E8 E5                PUSH H
 129    E9 21 0008           LXI H,8
 130    EC 39                DAD SP
 131    ED CD 0031'          CALL CCGINT##
 132    F0 EB                XCHG;;
 133    F1 21 0001           LXI H,1
 134    F4 CD 0000           CALL CCSUB##
 135    F7 D1                POP D
 136    F8 19                DAD D
 137    F9 CD 00E1'          CALL CCGCHAR##
 138    FC D1                POP D
 139    FD CD 0000           CALL CCXOR##
 140   100 D1                POP D
 141   101 7D                MOV A,L
 142   102 12                STAX D
 143   103 21 0001           LXI H,1
 144   106 E5                PUSH H
 145   107 21 0057           LXI H,87
 146   10A 39                DAD SP
 147   10B E5                PUSH H
 148   10C 21 0001           LXI H,1
 149   10F E5                PUSH H
 150   110 CD 0000           CALL write
 151   113 C1                POP B
 152   114 C1                POP B
 153   115 C1                POP B
 154   116 EB                XCHG;;
 155   117 21 0001           LXI H,1
 156   11A CD 0000           CALL CCNE##
 157   11D 7C                MOV A,H
 158   11E B5                ORA L
 159   11F CA 0137'          JZ CC9
 160   122 21 016D'          LXI H,CC2+16
 161   125 E5                PUSH H
 162   126 21 0002           LXI H,2
```

```
file: CPT.MAC    page:    4

line  loc ----object----  source

 163  129 E5              PUSH H
 164  12A CD 006B'        CALL fputs
 165  12D C1              POP B
 166  12E C1              POP B
 167  12F 21 0007         LXI H,7
 168  132 E5              PUSH H
 169  133 CD 0074'        CALL abort
 170  136 C1              POP B
 171  137                 CC9:
 172  137 21 0002         LXI H,2
 173  13A 39              DAD SP
 174  13B E5              PUSH H
 175  13C 21 0004         LXI H,4
 176  13F 39              DAD SP
 177  140 CD 00EE'        CALL CCGINT##
 178  143 EB              XCHG;;
 179  144 C1              POP B
 180  145 E1              POP H
 181  146 E5              PUSH H
 182  147 C5              PUSH B
 183  148 CD 0000         CALL CCDIV##
 184  14B EB              XCHG;;
 185  14C 11 0001         LXI D,1
 186  14F 19              DAD D
 187  150 D1              POP D
 188  151 CD 0080'        CALL CCPINT##
 189  154 C3 0082'        JMP CC4
 190  157                 CC5:
 191  157 21 0056         LXI H,86
 192  15A 39              DAD SP
 193  15B F9              SPHL
 194  15C C9              RET
 195  15D 75 73 61 67 65  CC2:DB 117,115,97,103,101,58,32,67,80,84
 195  162 3A 20 43 50 54
 196  167 20 6B 65 79 0A  DB 32,107,101,121,10,0,111,117,116,112
 196  16C 00 6F 75 74 70
 197  171 75 74 20 65 72  DB 117,116,32,101,114,114,111,114,10,0
 197  176 72 6F 72 0A 00
 198  17B                 EXT read
 199  17B                 EXT isatty
 200  17B                 EXT poll
 201  17B                 EXT auxbuf
 202  17B                 EXT abort
 203  17B                 EXT fputs
 204  17B                 EXT write
 205  17B                 EXT getarg
 206  17B                 EXT Ulink
 207  17B                 END
```

```
file: CPT.MAC    page:   5

 134' ABORT##          E' AUXBUF##          0' CC1:             15D' CC2:
  77' CC3:            82' CC4:            157' CC5:              D7' CC6:
  CC' CC7:            CF' CC8:            137' CC9:             149' CCDIV##
  BF' CCEQ##          FA' CCGCHAR##       141' CCGINT##         11B' CCNE##
  5B' CCOR##         152' CCPINT##         F5' CCSUB##           FE' CCXOR##
 12B' FPUTS##         35' GETARG##         AB' ISATTY##           0' MAIN::
  A3' POLL##          90' READ##            0  ULINK##          111' WRITE##
```

Damit ist die Assemblierung abgeschlossen. Sie finden nun auf der Diskette eine Datei mit dem Namen CPT.REL. Diese Datei enthält den erzeugten relokatierbaren Objektcode. Damit dieser als Programm laufen kann, muß er noch gelinkt werden. Linken bedeutet, daß alle für den Programmlauf benötigten Routinen aus der Bibliothek C.LIB hinzugefügt werden und dann ein unter CP/M aufrufbares Programm mit dem Namen CPT.COM daraus gemacht wird.

Kopieren Sie dazu nun die Datei CPT.REL auf die Diskette 3 (Linker) und geben Sie dann folgenden Befehl ein:

```
A>LNK -M CPT C.LIB
```

Der Linker meldet mit seiner Startmeldung und gibt dann auf dem Bildschirm aus, was er macht. Die Namen am Ende jeder Zeile sind die Namen der Programm-Module, die zum größten Teil aus der Bibliothek C.LIB hinzugebunden werden.

```
Small-Mac Linkage Editor, Version X.X (Rev. XX)
Copyright 1985 J. E. Hendrix

     17B Bytes at     0'   103 CPT
     3AC Bytes at   17B'   27E AUXBUF
      36 Bytes at   527'   62A AVAIL
     198 Bytes at   55D'   660 CALL
    1124 Bytes at   6F5'   7F8 CSYSLI
      53 Bytes at  1819'  191C EXIT
      8C Bytes at  186C'  196F FCLOSE
      14 Bytes at  18F8'  19FB FEOF
      B5 Bytes at  190C'  1A0F FFLUSH
      8B Bytes at  19C1'  1AC4 FGETC
      32 Bytes at  1A4C'  1B4F FPUTS
      A2 Bytes at  1A7E'  1B81 FREAD
      AD Bytes at  1B20'  1C23 FWRITE
      B1 Bytes at  1BCD'  1CD0 GETARG
       D Bytes at  1C7E'  1D81 ISATTY
      62 Bytes at  1C8B'  1D8E ISSPAC
       F Bytes at  1CED'  1DF0 MALLOC
      2F Bytes at  1CFC'  1DFF PAD
      81 Bytes at  1D2B'  1E2E POLL
      45 Bytes at  1DAC'  1EAF STRCHR
      60 Bytes at  1DF1'  1EF4 STRCMP
      93 Bytes at  1E51'  1F54 STRNCP
      3C Bytes at  1EE4'  1FE7 TOUPPE
     129 Bytes at  1F20'  2023 CSEEK
      90 Bytes at  2049'  214C FPUTC
       D Bytes at  20D9'  21DC END
Start In END
    20E6 Bytes (hex)
    8422 Bytes (dec)
```

Der Parameter -M beim Aufruf bedeutet wieder, daß der Linker bei der Arbeit Informationen über den Linkvorgang ausgeben soll. CPT ist der Name des zu linkenden Programms, auch hier wieder ohne Namenserweiterung REL. C.LIB ist die Bibliothek, in der der Linker nach im eigentlichen Programm nicht definierten Funktionen suchen soll. Nun ist das lauffähige Programm CPT.COM mit einer Größe von 8422 Bytes erzeugt worden. Bitte überzeugen Sie sich davon, indem Sie es nach den Anweisungen in Kapitel 5 auf eine Datei anwenden.

Kopieren Sie nun das Programm CPT.COM auf Ihre Anwendungsdiskette. Sie können dann die Dateien CPT.C, CPT.MAC, CPT.REL und CPT.COM von den Disketten 1 bis 3 herunterlöschen, damit wieder Platz für einen weiteren Kompiliervorgang vorhanden ist.

Diskettenaufteilung bei zwei Laufwerken

Wenn Sie an Ihrem Computer zwei Laufwerke angeschlossen haben, ist der Kompiliervorgang etwas einfacher. Sie können dann die Programme PIP.COM, CC.COM, MAC.COM und LNK.COM sowie die C-Bibliothek (C.LIB, C.NDX) auf der Diskette im Laufwerk A speichern, so daß das Laufwerk B für die Quelldateien (STDIO.H und CPT.C) verfügbar ist.

Ein zweites Beispiel

Damit die Vorgehensweise noch deutlicher wird, hier noch ein weiteres Beispiel. Es soll nun das Programm LST aus den Small-Tools kompiliert werden. Sehen Sie sich dazu das Programm einmal an. Wie Sie sehen können, werden bei diesem Programm außer STDIO.H noch vier weitere Dateien mit der Anweisung *#include* eingeschlossen. Diese werden natürlich bei der Kompilierung auch benötigt. Kopieren Sie also zunächst die Dateien LST.C, TOOLS.H, OUT.C, SAME.C und TRIM.C von der Diskette S1 auf die Diskette 1 (C-Compiler).

Rufen Sie nun den C-Compiler wieder mit dem Namen des Programms auf.

```
A>CC -M LST
```

Der Compiler zeigt wieder an, welche Funktionen er kompiliert. Nach der Beendigung haben Sie den Assemblercode in der Datei LST.MAC. Kopieren Sie diese auf die Diskette 2 (Small-Mac-Assembler). Rufen Sie den Assembler nun wieder ähnlich wie eben auf, geben Sie aber statt des Listingparameters -L den Parameter -NM an. Dieser Parameter bedeutet "keine Makros" ("No Macros"), der Compiler versteht dann also keine Makros mehr. Dies ist bei kompilierten C-Programmen auch nicht notwendig. Da-

für wird die Assemblierung dann um etwa 13% schneller. Der Listingparameter kann wegfallen, da ansonsten die Bildschirmausgabe die Assemblierung unnötig bremst. Geben Sie also ein:

```
A>MAC -NM LST
```

Der Assembler erzeugt die Datei LST.REL, die auf die Diskette 3 (Linker) kopiert werden muß. Hier rufen Sie dann mit folgendem Befehl den Linker auf:

```
A>LNK -M LST C.LIB
```

Wenn der Linker seine Arbeit beendet hat, befindet sich auf der Diskette das lauffähige Programm LST.COM.

Kompilierung der Small-Tools

Der Quellcode für jedes Small-Tools-Programm befindet sich in der Datei gleichen Namens aber mit der Erweiterung C. EDT ist in zwei Teile gegliedert, EDT.C und EDT2.C (die zur Kompilierzeit in EDT.C eingebunden wird). FMT besteht aus drei Teilen: FMT.C, FMT2.C, und FMT3.C. Die letzteren beiden werden zur Kompilierzeit in die Datei FMT.C eingebunden. Die restlichen C-Dateien sind allgemeine Funktionen, die in mehrere Small-Tools-Programme eingebunden werden.

Alle *#include*-Anweisungen in den Programmen gehen davon aus, daß die Include-Dateien auf dem Standardlaufwerk sind. Wenn man anders verfährt, muß man vor dem Kompilieren die *#include*-Anweisungen in den Programmen ändern. Man sollte sich also vor dem Kompilieren das zu kompilierende Programm anschauen und sicherstellen, daß sich alle Include-Dateien auf dem entsprechenden Laufwerk befinden. Ansonsten kann so vorgegangen werden, wie es oben beschrieben wurde. Nur bei den Programmen EDT und FNT tritt eine kleine Änderung auf. Da diese Programme recht groß sind, werden darin eine große Anzahl von Symbolen definiert, was dazu führt, das die Symboltabelle des Small-Mac-Assemblers überläuft. MAC muß also bei diesen beiden Programmen mit dem Schalter -S zur Vergrößerung der Symboltabelle aufgerufen werden, etwa so:

```
MAC -NM -S800 EDT
```

Änderung von Small-Tools-Parametern

Zur Kompilierzeit schließen alle Programme die Dateien STDIO.H und TOOLS.H ein. STDIO.H ist Bestandteil des Small-C-Compilers und sollte ohne Änderung bei jedem Small-C-Programm benutzt werden. TOOLS.H hingegen wird nur mit den Small-Tools-Programmen benutzt und kann an die entsprechenden Erfordernisse angepaßt werden. Darin werden unter anderem definiert:

1. die maximale Länge einer Textzeile (MAXLINE),
2. die Zeichenfolge, die den Bildschirm löscht (CLEAR),
3. die Größe des Bildschirms (CRT..) und der Druckseite (PTR..) und
4. welche Zeichen als Metazeichen dienen.

Änderungen in dieser Datei müssen vor dem Kompilieren gemacht werden. Die Tastatur und die Anwendung bei der Textverarbeitung sind wesentliche Kriterien für die einfache Benutzung der Metazeichen. Man sollte sorgfältig darüber nachdenken, wie man sie ändert und dann auch die Dokumentation entsprechend ändern.

```
/*
** Small-Tools definitions.
**
*/
#define MAXFN       15  /* max file name space */
#define EXTMARK     '.' /* file extension mark */
#define MAXLINE    192  /* max text line space */

        /* WY-50, TV-920, HZ-1500, AD-VP */
#define CLEAR "\33\53" /* screen erase */
#define CRTWIDE     80  /* screen width */
#define CRTHIGH     24  /* screen height */

#define PTRWIDE     80  /* page width */
#define PTRHIGH     66  /* page height */
#define PTRSKIP      8  /* page perforation skips */
#define PTRHDR       2  /* page header lines */

#define MAXPAT     257  /* max pattern in internal format */
#define CHAR       'c'  /* identifies a character */
#define BOL        '''  /* beginning of line */
#define EOL       '\''  /* end of line */
#define ANY        '?'  /* any character */
#define CCL        '['  /* begin character class */
#define NCCL       '~'  /* negation of character class */
#define CCLEND     ']'  /* end of character class */
#define CLOSURE    '*'  /* zero or more occurrences */
#define DITTO      '^'  /* whatever string matches pattern */
#define ESCAPE     ':'  /* escape character */
#define NOT        '~'  /* negation character */

#define DITCODE     -3
#define COUNT        1
#define PREVCL       2
#define START        3
#define CLOSIZE      4
```

Die Definition von CLEAR muß an den eigenen Computer angepaßt werden. CLEAR ist die Zeichenkette zum Löschen des Bildschirms und Positionierung des Cursors in die linke obere Ecke. In der gelieferten Version ist sie für die angegebenen Computer bzw. Terminals vorgesehen. Die Zeichen bedeuten ein Escape ("\33") und ein Plus ("\53"); die Zahlen sind oktale Werte. Wenn der Zielcomputer nun aber die Steuersequenzen des VT52-Terminals versteht, muß die Definition von CLEAR so lauten:

```
#define CLEAR "\33E\33H" /* screen erase */
```

Dies bedeutet beim VT52, daß der Bildschirm gelöscht wird (ESC E) und dann der Cursor in die linke obere Ecke positioniert wird (ESC H).

Die Zeilenzahl des Bildschirms wird mit CRTHIGH definiert. Meistens kann die Zahl 24 stehenbleiben, es gibt jedoch auch Microcomputer mit 25 Bildschirmzeilen. Hier sollte der Wert entsprechend geändert werden.

Die Definitionen, die mit PTR beginnen, beziehen sich auf den Drucker. PTRWIDE ist die Anzahl der Spalten pro Zeile. PTRHIGH ist die Anzahl Zeilen pro Seite; dies sollte auf jeden Fall auf die in Deutschland übliche Länge von 72 Zeilen abgeändert werden.

Es folgen noch mehrere Zeichendefinitionen für die Angabe von Sonderzeichen oder Zeichenklassen. Wenn der Zielcomputer den DIN-Zeichensatz verwendet, sind die Zeichencodes für "@[\]{|}~" identisch mit denen von "§ÄÖÜäöüß". Sollen also in den Parametern auch alle Umlaute verwendet werden können, müssen die Definitionen von CCL, NCCL, CCLEND und NOT auf jeden Fall geändert werden. Es empfehlen sich hier folgende Zeichen: '(' statt '[', ')' statt ']' und '#' statt '~'. Es ist zu beachten, daß sich dann die Bedienung aller Small-Tools entsprechend ändert.

Das Programm AR zur Verwaltung von Archivdateien

Der Quellcode des Small-C-Compilers und der Small-C-Bibliothek wird wegen der großen Anzahl der Module in sogenannten Archivdateien ausgeliefert. Bevor der Compiler oder ein Bibliotheksmodul kompiliert werden kann, muß es aus der jeweiligen Archivdatei in eine normale Datei herauskopiert werden. Der Quellcode des Small-C-Compilers befindet sich in der Archivdatei CC.ARC, der Quellcode der Bibliothek in CLIB.ARC.

Das Programm AR.COM dient dem Anlegen und der Verwaltung solcher Archivdateien. Damit können mehrere Textdateien in eine Archivdatei kopiert, einzelne Module wieder herausgezogen, neue hinzugefügt und alte ersetzt oder gelöscht werden. Ebenso ist es möglich, den Inhalt einer Archivdatei anzuzeigen. AR wird wie folgt bedient:

```
ar -{dptux} arcfile [file...]
```

Das erste Argument ist einer der angegebene Schalter. Das zweiter Argument ist der Name der Archivdatei. Als einzige Namenserweiterung ist ARC zugelassen. Sie muß angegeben werden. Die weiteren Parameter sind Datei- oder Modulnamen. Die Schalter haben folgende Bedeutung:

`-d`	Löscht die angegebenen Module aus der Archivdatei.
`-p`	Gibt die angegebenen oder alle (ohne Namen) Module auf der Standardausgabe aus.
`-t`	Gibt eine Liste der Module in der Archivdatei auf der Standardausgabe aus.
`-u`	Aktualisiert die Archivdatei indem die genannten Module hinzugefügt oder durch neue ersetzt werden. Dieser Schalter wird auch zum Anlegen einer neuen Archivdatei verwendet. Wenn keine Dateinamen angegeben wurden, werden sie von der Standardeingabe gelesen.
`-x`	Die angegebenen oder alle (ohne Namen) Module werden aus der Archivdatei heraus in eigene Dateien kopiert.

Beispiele

BEFEHL BESCHREIBUNG

`AR -T CLIB.ARC`
Zeigt eine Liste aller Module in der Archivdatei CLIB.ARC an.

`AR -P CLIB.ARC ATOIB.C`
Gibt das Modul ATOIB.C aus der Archivdatei CLIB.ARC auf der Standardausgabe aus.

`AR -X CC.ARC CC1.C`
: Kopiert das Modul CC1.C aus der Archivdatei CC.ARC in eine eigene Datei.

`AR -X CC.ARC`
: Kopiert alle Module aus der Archivdatei CC.ARC in eigene Dateien.

`AR -D CLIB.ARC ATOIB.C`
: Löscht das Modul ATOIB.C aus der Archivdatei CLIB.ARC.

`AR -U CC.ARC CC.DEF`
: Bringt die Datei CC.DEF in die Archivdatei CC.ARC. Wenn das Modul CC.DEF darin bereits existiert, wird die alte Version durch die neue ersetzt.

Meldungen

MELDUNG ERLÄUTERUNG

`copied new xxxxxxxx`
: Die genannte Datei wurde in die Archivdatei aufgenommen.

`printed xxxxxxxx`
: Die angegebene Datei ist ausgedruckt worden.

`created xxxxxxxx`
: Die angegebene Datei ist angelegt worden.

`dropped old %s`
: Die alte Version eines Moduls wurde durch eine neue Version überschrieben.

`file - xxxxxxxx`
: Die angegebene Datei wird gerade verarbeitet.

Fehlermeldungen

MELDUNG ERLÄUTERUNG

`xxxxxxxx: can't open`
: Eine Datei kann nicht geöffnet werden.

`fatal errors - archive not altered`
: Es ist ein nicht behebbarer Fehler aufgetreten, die Archivdatei wurde nicht geändert.

`can't rename xxxxxxxx to xxxxxxxx`
: Eine Datei kann nicht umbenannt werden. Wahrscheinlich befindet sich auf der Diskette bereits eine Datei mit dem neuen Namen.

`delete by name only`
: Bei der Option -d muß mindestens ein Modulname angegeben werden.

`xxxxxxxx: can't create`
Eine Ausgabedatei für ein Modul, das aus der Archivdatei extrahiert werden soll, kann nicht geöffnet werden. Wahrscheinlich ist das Diskettenverzeichnis voll.

`too many file names`
Es sind mehr als MAXFILES Dateinamen angegeben worden. Entweder weniger Dateien angeben oder AR.C mit einem höheren Wert für MAXFILES neu kompilieren.

`xxxxxxxx: duplicate file names`
Eine Datei wurde in der Befehlszeile mehrmals genannt.

`archive not in proper format`
Die angegebene Archivdatei hat kein gültiges Format. Entweder ist die Datei keine Archivdatei oder sie wurde durch einen Fehler zerstört.

`xxxxxxxx not in archive`
Das genannte Modul befindet sich nicht in der Archivdatei.

Kompilierung des Small-C-Compilers

Das wichtigste am Small-C-Compiler ist, das er selbst in Small-C geschrieben ist und mit Quellcode ausgeliefert wird. Der Compiler ist selbst nur ein einfaches Small-C-Programm und kann deshalb verwendet werden, um neue Versionen von sich selbst zu erzeugen. Die Kompilierung des Small-C-Compilers unterscheidet sich im wesentlichen nicht vom Kompilieren eines beliebigen anderen C-Programms.

Der Compiler ist in vier Module aufgeteilt, das jedes für sich kompiliert wird. Alle vier Module werden dann mit dem Linker zu einem lauffähigen Programm zusammengebunden. Alternativ kann der Small-C-Compiler auch in einem Zuge kompiliert werden, dazu ist jedoch ausreichend Speicher erforderlich.

Die Module des Small-C-Compilers befinden sich in der Archivdatei CC.ARC und können mit dem Programm AR herauskopiert werden. Die Bedienung von AR ist im vorhergehenden Abschnitt beschrieben.

Die Quelldateien fallen in drei Kategorien:

1. Die Datei CC.DEF enthält alle *#define*-Anweisungen für den Compiler. Die meisten dieser Definitionen sind Symbole für Konstanten, einige steuern jedoch auch die Kompilierung. Diese werden in Verbindung mit den Präprozessoranweisungen *#ifdef*, *#ifndef*, *#else* und *#endif* verwendet, um bestimmte Zeilen in die Kompilierung mit einzubeziehen oder davon auszuschließen. Sie beeinflussen also, welche Eigenschaften im neuem Compiler enthalten sind. Eines dieser Symbole, SEPARATE, hat keinen Einfluß auf den erzeugten Compiler, sondern bewirkt, daß der Compiler in einzelnen Modulen kompiliert werden kann.
2. Die Dateien CC1.C, CC2.C, CC3.C und CC4.C enthalten die Hauptteile des Compilers. CC1.C enthält die Deklarationen der globalen Objekte und die anderen Dateien enthalten externe Deklarationen derselben Objekte. Diese Deklarationen werden nur kompiliert, wenn SEPARATE definiert ist. Jede Hauptdatei enthält auch eine *#include*-Anweisung für STDIO.H. Schließlich enthält jeder Hauptteil noch #include-Anweisungen für die weiteren Module des jeweiligen Compilerteils. CC1.C enthält zusätzlich noch weitere *#include*-Anweisungen für die Teile 2, 3 und 4. Diese Anweisungen werden jedoch nur ausgeführt, wenn SEPARATE nicht definiert ist, der Compiler also in einem Zuge kompiliert wird. In diesem Fall braucht nur CC1 kompiliert werden, sonst muß jeder Teil (CC1, CC2, CC3 und CC4) getrennt kompiliert werden. NOTICE.H enthält Copyright und Versionsnummer.

3. Die Dateien mit zwei Ziffern im Dateinamen sind Unterdateien der jeweiligen Hauptteile. Die erste Ziffer kennzeichnet den Hauptteil, zu dem die Datei gehört, die zweite Ziffer ist eine fortlaufende Nummer. Die folgende Tabelle zeigt die Verbindung der Dateien untereinander:

Teil	Haupt-datei	Schließt folgende Dateien ein					
1	CC1.C	STDIO.H	CC.DEF	CC11.C	CC12.C	CC13.C	NOTICE.H
2	CC2.C	STDIO.H	CC.DEF	CC21.C	CC22.C		
3	CC3.C	STDIO.H	CC.DEF	CC31.C	CC32.C	CC33.C	
4	CC4.C	STDIO.H	CC.DEF	CC41.C	CC42.C		

Um verschiedene Compiler-Optionen zu kontrollieren, sind mehrere Symbole in der Datei CC.DEF definiert worden. Die wichtigsten werden im folgenden erläutert.

Das Symbol DYNAMIC kompiliert Anweisungen, die dynamisch Speicher für verschiedene Tabellen und Arrays innerhalb des Compilers anlegen. Wenn DYNAMIC nicht definiert ist, werden die Tabellen und Arrays direkt im Compiler angelegt. Dieses Symbol kontrolliert auch Anweisungen, die CCAVAIL aufrufen, deren primärer Zweck ist, den verfügbaren freien Speicherplatz zurückzugeben. Es wird aber benutzt, um sicherzustellen, daß sich Stack und zugeordneter Speicher nicht überlappen. Wenn serielle Tabellensuche zusammen mit dynamischer Speicherzuordnung benutzt wird, wird jede neue Eintragung in der globalen Symboltabelle separat zugeordnet. Diese Tabelle kann wachsen, bis sie den Maschinenstack überlappt und dann einen Zuordnungsfehler produziert.

LINK setzt voraus, daß die Compilerausgabe mit einem verschiebbaren Assembler und einem Linker weiterverarbeitet wird. Als extern deklarierte globale Variablen werden als externe Referenzen und andere globale Variablen als Einsprungspunkte kompiliert. In diesem Fall können mehrteilige Programme nicht während des Assemblierens kombiniert werden und die Startlabel-Option (-B# Schalter) ist nicht verfügbar.

Durch die Definition von COL werden Label in der Ausgabe durch einen Doppelpunkt beendet.

Wenn UPPER definiert ist, werden Symbole in Großbuchstaben in die Symboltabelle eingetragen. Wenn das der Assembler nicht erfordert, sollte die Definition von UPPER ausgeschaltet werden.

NOCCARGC ist eine Laufzeit-Option, die dem Compiler sagt, daß er keinen Code zur Übergabe der Zahl der Argumente erzeugen soll. Als Ergebnis erhält man kleinere, schnellere Programme, wenn bekannt ist, daß kein Aufruf zur Laufzeitroutine CCARGC erfolgt.

SEPARATE setzt voraus, daß der Compiler in Teilen kompiliert werden soll, anstatt in einem. In diesem Fall müssen die Dateien CC1.C, CC2.C, CC3.C, und CC4.C getrennt kompiliert werden. Diese wiederum enthalten untergeordnete Dateien; zum Beispiel CC11.C, CC12.C und CC13.C (für Teil 1). Wenn dieses Symbol fehlt, dann schließt die Datei CC1.C alle untergeordneten Dateien ein und CC2.C, CC3.C und CC4.C werden nicht benutzt.

Vier Symbole erlauben es zu bestimmen, welche Sprachanweisungen durch den neuen Compiler unterstützt werden. Man kann sie auslassen, um den Compiler so klein zu machen, daß er selbst komplett auf einer Maschine mit weniger als 56 KByte kompiliert werden kann.

STDO kontrolliert die *do*-Anweisung. STFOR kontrolliert die *for*-Anweisung. STSWITCH kontrolliert die Anweisungen *switch*, *case* und *default*. STGOTO kontrolliert die Anweisung *goto*. Diese Definitionen verbinden mit jeder Anweisung auch einen numerischen Wert; der Compiler benutzt dies, um zu entscheiden, ob die letzte Anweisung in einer Funktion ein *return* ist.

Durch die Definition von OPTIMIZE wird die Peephole-Optimierung eingeschlossen. Die Optimierungstechniken werden im folgenden für interessierte Anwender beschrieben.

Beschreibung der Codeoptimierung

Maschinenunabhängige Optimierung wird durch eine Änderung des Ausdrucksanalysierers, maschinenabhängige Optimierung durch eine wahlweisen Ausgabeoptimierung (*peephole*) erreicht.

Die maschinenunabhängige Optimierung benutzt folgende Techniken:

1. Ausdrücke oder Teilausdrücke, die als Ergebnis einen konstanten Wert haben, erzeugen nur eine einzigen direkten Ladebefehl.
2. Nachdem der Code auf der rechten Seite eines binären Befehls erzeugt wurde, wird das vorsichtshalber vorgenommene *push/pop* der rechten Seite durch einen Tausch ersetzt, wenn das Zweitregister nicht benutzt worden war. Wenn aber der Wert auf der linken Seite eine Konstante ist, wird sie stattdessen sofort in das Zweitregister geladen.
3. Konstanten, die von Integerzeigern oder Arrays addiert oder subtrahiert werden, werden bereits durch den Compiler verdoppelt und nicht erst während der Ausführung.
4. Für die Anweisungen *if(const)*, *while(const)*, und *for(...; const; ...)* wird kein geschachtelter Code erzeugt. Der Compiler kümmert sich nicht um die Löschung dieser Codesequenzen, da es sich wahrschein-

lich um einen Programmfehler handelt; *#ifdef* und *#ifndef* sollten verwendet werden, um bedingten Code während des Kompilierens zu entfernen.

5. Prüfung auf Null (zum Beispiel *while(i >= 0)* oder *if(abc() == 0)* und so weiter) ergibt eine besondere Codefolge, die kleiner und schneller ist, als die übliche Prozedur Null zu laden, eine Bibliotheksroutine aufzurufen und dann den zurückgegebenen Wert auf 0 oder 1 zu testen.
6. Nullwerte erzeugen keinen Code zur Addition mit Arrayadressen und Zeigerwerten.
7. Lokale Variablen werden alle zusammen zur gleichen Zeit zugeordnet, wenn die erste ausführbare Anweisung angetroffen wird. Deklarationen nach diesem Punkt sind nicht erlaubt, außer innerhalb innerer Blöcke.
8. Unnötige Sprünge um Anweisungen, die durch *else* kontrolliert werden, werden vermieden. Dies ist der Fall, wenn ein *return* oder *goto* dem *else* vorausgeht.
9. Die Funktion *modstk* erzeugt jetzt zwei Vertauschungen, um das Erstregister nur bei einem *return* mit einem Wert zu erhalten.

Maschinenabhängige Optimierung wird durch die zwei Funktionen, *putstk* und *peephole*, erreicht. *Putstk* erzeugt jetzt anstatt CALL CCPCHAR die Befehlsfolge MOV A,L/STAX D. *Peephole* ist die schon erwähnte Ausgabeoptimierung. Für den durch einen Ausdruck erzeugten Code wird ein Puffer benutzt. Wenn der Puffer zurückgeschrieben wird, untersucht *peephole()* die Ausgabe und macht passende Änderungen. Die Ausgabeoptimierung ist eine Compiler-Option, die manche für ein getrenntes Hilfsprogramm besser geeignet finden. Ich fand es jedoch unwiderstehlich, diese einfache Funktion dem Compiler huckepack aufzuladen; man kann dadurch einfacher und leichter optimiert kompilieren.

Peephole() benutzt zwei Techniken. Erstens werden Integer von der Spitze des Stacks mit einer POP H/PUSH H Sequenz geholt anstatt mit der üblichen Sequenz LXI H,0/DAD SP/CALL CCGINT. Integer direkt unterhalb der Stack-Spitze werden mit der Sequenz POP B/POP H/PUSH H/PUSH B gelesen. Wenn ein XCHG dem Lesezyklus folgt, wird der gewünschte Operand direkt in das DE-Registerpaar geladen. Dadurch erhält man schnelleren und kompakteren Code; dies ist immer wirksam, wenn der Compiler *peephole()* enthält.

Die zweite Technik ersetzt oft benutzte Befehlssequenzen durch neue Einsprungspunkte in die Laufzeitbibliothek. Diese Technik reduziert die Programmgröße auf Kosten der Geschwindigkeit; sie muß bei der Laufzeit angefordert werden (-o), um wirksam zu sein.

Übergabe der Argumentzahl

Wenn eine Funktion aufgerufen wird, wird die Zahl der zu übergebenden Argument in den Akkumulator gestellt. Dies benötigt nur zwei Bytes. Um die Zahl zu holen, weist die aufgerufene Funktion einer Variablen einfach den von der Funktion CCARGC (Großbuchstaben) zurückgegeben Wert zu. Dies muß als erstes in der Funktion gemacht werden, da andere Operationen Laufzeitbibliotheken aufrufen können, die den Akkumulator zerstören. CCARGC ist ein neuer Einsprungspunkt in der Laufzeitbibliothek; sie definiert einfach CCSXT neu, die A mit Vorzeichen nach HL bringt. Damit hat man 127 Argumente, bevor das Programm abstürzt. Aus offensichtlichen Gründen erzeugt der Compiler keinen Code, um die Anzahl der Argument zu laden, wenn CCARGC aufgerufen wird. Da viele Programme die Zahl der Argumente nicht übergeben, übergeht der Compiler dies in Programmen, die die Anweisung *#define NOCCARGC* (Großbuchstaben) enthalten. Dies reduziert Prpgrammgröße und Ausführungszeit.

Kompilierung der Small-C-Bibliothek

Die Module der Small-C-Bibliothek befinden sich in der Archivdatei CCLIB.ARC und können mit dem Programm AR herauskopiert werden. Die Bedienung von AR ist in einem vorhergehenden Abschnitt beschrieben.

Für die Kompilierung auf Systemen mit wenig Speicherplatz ist es am günstigsten, wenn das gewünschte Modul einzeln aus der Archivdatei herauskopiert und bearbeitet wird. Wenn alle Änderungen korrekt durchgeführt worden sind, muß das nach Kompilierung und Assemblierung entstandene REL-Modul mit LIB in die Bibliothek C.LIB eingebunden werden, damit es wirksam wird (bei Verwendung des Microsoft-Assemblers in die Bibliothek CLIB.REL). Das fertige Quellmodul sollte mit AR wieder in die Archivdatei CLIB.ARC aufgenommen werden.

Folgende Module befinden sich in der Archivdatei CLIB.ARC:

```
ABS.C         FSCANF.C      MALLOC.C
ATOI.C        FWRITE.C      OTOI.C
ATOIB.C       GETARG.C      PAD.C
AUXBUF.C      GETCHAR.C     POLL.C
AVAIL.C       ISALNUM.C     PUTCHAR.C
CALL.MAC      ISALPHA.C     PUTS.C
CALLOC.C      ISASCII.C     RENAME.C
CLEARERR.C    ISATTY.C      REVERSE.C
CLIB.DEF      ISCNTRL.C     REWIND.C
CSEEK.C       ISCONS.C      SIGN.C
CSYSLIB.C     ISDIGIT.C     STDIO.H
CTELL.C       ISGRAPH.C     STRCAT.C
DTOI.C        ISLOWER.C     STRCHR.C
EXIT.C        ISPRINT.C     STRCMP.C
FCLOSE.C      ISPUNCT.C     STRCPY.C
FEOF.C        ISSPACE.C     STRLEN.C
FERROR.C      ISUPPER.C     STRNCAT.C
FFLUSH.C      ISXDIGIT.C    STRNCMP.C
FGETC.C       ITOA.C        STRNCPY.C
FGETS.C       ITOAB.C       STRRCHR.C
FOPEN.C       ITOD.C        TOASCII.C
FPRINTF.C     ITOO.C        TOLOWER.C
FPUTC.C       ITOU.C        TOUPPER.C
FPUTS.C       ITOX.C        UNGETC.C
FREAD.C       LEFT.C        UNLINK.C
FREE.C        LEXCMP.C      UTOI.C
FREOPEN.C     LINK.MAC      XTOI.C
```

Kompilierung der Small-Mac-Programme

Bei den Small-Mac-Programmen wird weniger mit Include-Dateien gearbeitet als beim Compiler oder den Tools. Dafür befinden sich mehrere Module in der Bibliothek M.LIB. Dies sind die Module, die von allen oder mehreren Programmen benötigt werden. Jedes Programm muß also nach dem Kompilieren außer mit C.LIB auch mit M.LIB gelinkt werden. Dabei ist es wichtig, daß M.LIB beim Aufruf des Linkers zuerst genannt wird, da darin Routinen aus C.LIB verwendet werden, die ansonsten beim Durchsuchen von C.LIB noch nicht definiert wären. Der Aufruf sieht so aus:

```
LNK -M PRG M.LIB C.LIB
```

Hier eine Übersicht darüber, welche Dateien mit *#include* in welche Programme eingeschlossen werden (STDIO.H ist nicht berücksichtigt):

Programme:

CMIT.C	NOTICE.H	MAC.H	MIT.H		
DREL.C	NOTICE.H	MAC.H		REL.H	
LGO.C	NOTICE.H				
LIB.C	NOTICE.H			REL.H	
LNK.C	NOTICE.H			REL.H	
MAC.C	NOTICE.H	MAC.H	MIT.H	REL.H	
MAC2.C		MAC.H		REL.H	EXT.H
MAC3.C		MAC.H		REL.H	EXT.H

Bibliotheksmodule:

EXTEND.C	MAC.H	
FILE.C		
GETREL.C	MAC.H	REL.H
INT.C		
MESS.C		
MIT.C	MAC.H	
PUTREL.C	MAC.H	REL.H
REL.C		
REQ.C		
SCAN.C	MAC.H	
SEEREL.C		REL.H
WAIT.C		

Bei Änderungen an Bibliotheksmodulen müssen die entstehenden REL-Dateien mit LIB in die Bibliothek M.LIB gebracht werden; die alten Versionen der Module werden dabei überschrieben.

Die Kompilierung der Programme CMIT, DREL, LGO, LIB und LNK geht so vor sich, wie zu Beginn dieses Kapitels an anderen Beispielen beschrieben. Die Kompilierung des Small-Mac-Makorassemblers MAC ist etwas anders. Hier werden die Teile MAC, MAC2 und MAC3 seaparat kompiliert und anschließend mit dem Linker zusammengebunden. Die entstandene COM-Datei (MAC.COM) muß mit CMIT noch an den Befehlssatz des 8080 oder Z80 angepaßt werden (CMIT mit dem Schalter -C aufrufen). Dazu muß sich auf der Diskette auch die Datei 8080.MIT oder Z80.MIT befinden. Erst dann ist der Makroassembler lauffähig.

Anhang A: Patch von CP/M 2.2 für Kleinbuchstaben

Mehrere Programme von Small-Tools verlangen in der Befehlszeile Textmuster als Argumente. Unglücklicherweise wandelt der CCP von CP/M alle Kleinbuchstaben in Großbuchstaben um und macht damit alle Versuche zunichte, Kleinbuchstaben anzugeben. Dies gilt genauso für SUBMIT.COM.

Mit den folgenden Patches kann man alle Möglichkeiten der Small-Tools-Programme voll nutzen, außerdem kann man dann Dateinamen mit Kleinbuchstaben angeben. Bei Small-C-Programmen werden die Dateinamen immer in Großbuchstaben umgewandelt, PIP erwartet jedoch, daß dies der CCP macht. Gelegentlich gibt es deshalb bei PIP Ärger mit den Kleinbuchstaben. Deswegen und weil Laufwerksangaben und eingebaute Befehle in Großbuchstaben erscheinen müssen, ist es am besten, wenn man die Tastatur auf Großbuchstaben einstellt (CAPS LOCK), außer wenn man Textmuster eingeben oder mit dem Editor arbeiten will. Schalter, die an die Small-Tools-Programme übergeben werden sollen, können entweder in Klein- oder in Großbuchstaben angegeben werden.

Die Patches sind hier in der Version für den Schneider CPC 664 beschrieben, die genannten Adressen können bei anderen Systemen abweichen.

Patch des CCP für Kleinbuchstaben mit MOVCPM, DDT und SYSGEN.

Zunächst müssen mit MOVCPM die Betriebssystemspuren in eine Datei kopiert werden. Beim Schneider CPC 664 erfolgt dies mit:

```
MOVCPM 179 *
SAVE 34 CPM44.COM
```

Das Betriebssystem steht nun in der Datei CPM44.COM. Diese Datei wird nun mit DDT geladen und nach folgender Befehlsfolge durchsucht:

```
0AB0  CP    61
0AB2  RC
0AB3  CPI   7B
0AB5  RNC
0AB6  ANI   5F   <---
0AB8  RET
```

Es muß der Wert 5F bei AB7h in FF geändert und die Datei mit dem oben genannten SAVE-Befehl wieder auf die Diskette geschrieben werden. Mit SYSGEN kann das CP/M dann wieder auf die Systemspuren zurückgeschrieben werden:

```
SYSGEN CPM44.COM
```

Patch von SUBMIT.COM für Kleinbuchstaben.

SUBMIT.COM wird mit DDT geladen und nach dem folgenden Code durchsucht:

```
0362  LDA  0675
0365  SUI  61
0367  CPI  1A
0369  JNC  0374
036C  LDA  0675
036F  ANI  5F   <---
0371  STA  0675
0374  LDA  0675
0377  RET
```

Der Wert 5F bei 370h muß in FF geändert werden. Dann verläßt man DDT und speichert das geänderte SUBMIT mit folgendem Befehl auf der Diskette:

```
SAVE 5 SUBMIT.COM
```

Anhang B: Bedienungshinweise der Programme

```
usage: CC    [file]... [-M] [-A] [-P] [-L#] [-O] [-B#]
usage: AR    -{dptux} arcfile [file...]

usage: MAC   [-L] [-NM] [-P] [-S#] [object] source...
usage: LNK   [-B] [-G] [-M] program [module/library...]
usage: LGO   [-G] [-M] program
usage: LIB   -{DPTUX}[A] library [module...]
usage: CMIT  [-C] [-L] [table] [mac]
usage: DREL

usage: CHG   pattern [replacement]
usage: CNT   [file] [-C|-W|-L]
usage: CPY   [file]... [.?] [-B] [-NCR] [-NLF] [-T#,#]
usage: CPT   key
usage: DTB   [#]... [+#]
usage: EDT   [file] [-V]
usage: ETB   [#]... [+#]
usage: FND   [~]pattern
usage: FNT   [device]
usage: FMT   [mergefile] [-BC#] [-EC#] [-BP#] [-EP#]
             [-PO#] [-NP] [-T] [-I] [-U] [-S] [-BS#] [-NR]
usage: LST   [file] [-C#] [-PW#] [-PL#] [-NB] [-NN] [-NP]
usage: MRG   file [file] [-1|-2|-3|-F]
usage: PRT   [file].. [.?] [-NN] [-NH|-NS] [-LM#] [-BP#] [-EP#] [-P] [-NR]
usage: SRT   [-C#|-F#?] [-D] [-U] [-Tx] [-Q]
usage: TRN   [~]from [to]
```

Anhang C: Übersicht aller Small-C-Funktionen

Die Parameter der Funktionen in der folgenden Übersicht haben folgende Bedeutung:

Parameter	englisch	deutsch
abort		Abbruchcode
addr	address	Speicheradresse
arg1	argument 1	Argument 1
arg2	argument 2	Argument 2
argc	argument count	Argumentzähler
argv	argument variables	Argumentvariablen
base		Basis
c	character	Zeichen
c1	character 1	Zeichen 1
c2	character 2	Zeichen 2
ch	character	Zeichen
cnt	count	Zähler
dest	destination	Zielstring
errcode	error code	Fehlercode
fd	file descriptor	Dateideskriptor
mode		Modus
n	number	Anzahl
name		Dateiname
nbr	number	Zahl
new		Neuer Dateiname
old		Alter Dateiname
pause		Pause
ptr	pointer	Zeiger
size		Größe
sour	source	Quellstring
str	string	String
str1	string 1	String 1
str2	string 2	String 2
sz	size	Größe

```
abs(nbr) int nbr;
atoi(str) char *str;
atoib(str, base) char *str; int base;
auxbuf(fd, size) int fd, size;
avail(abort) int abort;
calloc(nbr, sz) int nbr, sz;
clearerr(fd) int fd;
cseek(fd, offset, from) int fd, offset, from;
ctell(fd) int fd;
dtoi(str, nbr) char *str; int *nbr;
exit(errcode) int errcode; (alias abort)
fclose(fd) int fd;
feof(fd) int fd;
ferror(fd) int fd;
fflush(fd) int fd;
fgetc(fd) int fd; (alias getc)
fgets(str, sz, fd) char *str; int sz, fd;
fopen(name, mode) char *name, *mode;
fprintf(fd,str,arg1,arg2, ...) int fd; char *str;
fputc(c, fd) char c; int fd; (alias putc)
fputs(str, fd) char *str; int fd;
fread(ptr,sz,cnt,fd) char *ptr; int sz, cnt, fd;
free(addr) char *addr; (alias cfree)
freopen(name, mode, fd) char *name, *mode; int fd;
fscanf(fd, str, arg1, arg2, ...) int fd; char *str;
fwrite(ptr,sz,cnt,fd) char *ptr; int sz, cnt, fd;
getarg(nbr,str,sz,argc,argv)
       char *str; int nbr, sz, argc, *argv;
getchar()
gets(str) char *str;
isalnum(c) char c;
isalpha(c) char c;
isascii(c) char c;
isatty(fd) int fd;
iscntrl(c) char c;
iscons(fd) int fd;
isdigit(c) char c;
isgraph(c) char c;
islower(c) char c;
isprint(c) char c;
ispunct(c) char c;
isspace(c) char c;
isupper(c) char c;
isxdigit(c) char c;
```

```
itoa(nbr, str) int nbr; char *str;
itoab(nbr, str, base) int nbr; char *str; int base;
itod(nbr, str, sz) int nbr, sz; char *str;
itoo(nbr, str, sz) int nbr, sz; char *str;
itou(nbr, str, sz) int nbr, sz; char *str;
itox(nbr, str, sz) int nbr, sz; char *str;
left(str) char *str;
lexcmp(str1, str2) char *str1, *str2;
lexorder(c1, c2) char c1, c2;
malloc(nbr) int nbr;
otoi(str, nbr) char *str; int *nbr;
pad(str, ch, n) char *str, ch; int n;
poll(pause) int pause;
printf(str, arg1, arg2, ...) char *str;
putchar(c) char c;
puts(str) char *str;
read(fd, ptr, cnt) int fd, cnt; char *ptr;
rename(old, new) char *old, *new;
reverse(str) char *str;
rewind(fd) int fd;
scanf(str, arg1, arg2, ...) char *str;
sign(nbr) int nbr;
strcat(dest, sour) char *dest, *sour;
strchr(str, c) char *str, c;
strcmp(str1, str2) char *str1, *str2;
strcpy(dest, sour) char *dest, *sour;
strlen(str) char *str;
strncat(dest, sour, n) char *dest, *sour; int n;
strncmp(str1, str2, n) char *str1, *str2; int n;
strncpy(dest, sour, n) char *dest, *sour; int n;
strrchr(str, c) char *str, c;
toascii(c) char c;
tolower(c) char c;
toupper(c) char c;
ungetc(c, fd) char c; int fd;
unlink(name) char *name; (alias delete)
utoi(str, nbr) char *str; int *nbr;
write(fd, ptr, cnt) int fd, cnt; char *ptr;
xtoi(str, nbr) char *str; int *nbr;
```

Anhang D: Maschinen-Instruktions-Tabellen

Maschinen-Instruktions-Tabelle für 8080

```
CE_x1 ACI x
88¯   ADC B|C|D|E|H|L|M|A
80    ADD B|C|D|E|H|L|M|A
C6_x1 ADI x
A0¯   ANA B|C|D|E|H|L|M|A
E6_x1 ANI x

CD_x2 CALL x
DC¯x2 CC x
FC¯x2 CM x
2F¯   CMA
3F    CMC
B8    CMP B|C|D|E|H|L|M|A
D4_x2 CNC x
C4¯x2 CNZ x
F4¯x2 CP x
EC¯x2 CPE x
FE¯x1 CPI x
E4¯x2 CPO x
CC¯x2 CZ x

27    DAA
39    DAD SP
19    DAD D
29    DAD H
09    DAD B
3D    DCR A
05    DCR B
0D    DCR C
15    DCR D
1D    DCR E
25    DCR H
2D    DCR L
35    DCR M
0B    DCX B
1B    DCX D
2B    DCX H
3B    DCX SP
F3    DI

FB    EI
76    HLT

DB_x1 IN x
3C¯   INR A
04    INR B
0C    INR C
14    INR D
1C    INR E
24    INR H
2C    INR L
34    INR M
03    INX B
13    INX D
23    INX H
33    INX SP
```

```
DA_x2 JC x
FA_x2 JM x
C3_x2 JMP x
D2_x2 JNC x
C2_x2 JNZ x
F2_x2 JP x
EA_x2 JPE x
E2_x2 JPO x
CA_x2 JZ x

3A_x2 LDA x
0A    LDAX B
1A    LDAX D
2A_x2 LHLD x
21_x2 LXI H,x
11_x2 LXI D,x
31_x2 LXI SP,x
01_x2 LXI B,x

7C    MOV A,H|A,L|A,M|A,A
54    MOV D,H|D,L|D,M|D,A
5D    MOV E,L|E,M|E,A
78    MOV A,B|A,C|A,D|A,E
40    MOV B,B|B,C|B,D|B,E|B,H|B,L|B,M|B,A
48    MOV C,B|C,C|C,D|C,E|C,H|C,L|C,M|C,A
50    MOV D,B|D,C|D,D|D,E
58    MOV E,B|E,C|E,D|E,E|E,H
60    MOV H,B|H,C|H,D|H,E|H,H|H,L|H,M|H,A
68    MOV L,B|L,C|L,D|L,E|L,H|L,L|L,M|L,A
77    MOV M,A
70    MOV M,B|M,C|M,D|M,E|M,H|M,L
3E_x1 MVI A,x
06_x1 MVI B,x
0E_x1 MVI C,x
16_x1 MVI D,x
1E_x1 MVI E,x
26_x1 MVI H,x
2E_x1 MVI L,x
36_x1 MVI M,x

00    NOP

B5    ORA L|M|A
B0    ORA B|C|D|E|H
F6_x1 ORI x
D3_x1 OUT x

E9    PCHL
C1    POP B
D1    POP D
E1    POP H
F1    POP PSW
E5    PUSH H
D5    PUSH D
C5    PUSH B
F5    PUSH PSW

17    RAL
1F    RAR
D8    RC
C9    RET
20    RIM
07    RLC
F8    RM
D0    RNC
C0    RNZ
F0    RP
E8    RPE
E0    RPO
```

```
0F     RRC
C7     RST 0
D7     RST 16
DF     RST 24
E7     RST 32
EF     RST 40
F7     RST 48
FF     RST 56
CF     RST 8
C8     RZ

98     SBB B|C|D|E|H|L|M|A
DE_x1  SBI x
22_x2  SHLD x
30     SIM
F9     SPHL
32_x2  STA x
02     STAX B
12     STAX D
37     STC
90     SUB B|C|D|E|H|L|M|A
D6_x1  SUI x

EB     XCHG
A8     XRA B|C|D|E|H|L|M|A
EE_x1  XRI x
E3     XTHL
```

Maschinen-Instruktions-Tabelle für Z80

```
DD_8E_x1     ADC A,(IX+x)
FD_8E_x1     ADC A,(IY+x)
88_   _      ADC A,B|A,C|A,D|A,E|A,H|A,L|A,(HL)|A,A
CE_x1        ADC A,x
ED_4A        ADC HL,BC
ED_5A        ADC HL,DE
ED_6A        ADC HL,HL
ED_7A        ADC HL,SP
DD_86_x1     ADD A,(IX+x)
FD_86_x1     ADD A,(IY+x)
80_   _      ADD A,B|A,C|A,D|A,E|A,H|A,L|A,(HL)|A,A
C6_x1        ADD A,x
09_          ADD HL,BC
19           ADD HL,DE
29           ADD HL,HL
39           ADD HL,SP
DD_09        ADD IX,BC
DD_19        ADD IX,DE
DD_29        ADD IX,IX
DD_39        ADD IX,SP
FD_09        ADD IY,BC
FD_19        ADD IY,DE
FD_29        ADD IY,IY
FD_39        ADD IY,SP
DD_A6_x1     AND (IX+x)
FD_A6_x1     AND (IY+x)
A0_          AND B|C|D|E|H|L|(HL)|A
E6_x1        AND x

DD_CB_x1_46  BIT 0,(IX+x)
FD_CB_x1_46  BIT 0,(IY+x)
CB_40_   _   BIT 0,B|0,C|0,D|0,E|0,H|0,L|0,(HL)|0,A
DD_CB_x1_4E  BIT 1,(IX+x)
FD_CB_x1_4E  BIT 1,(IY+x)
CB_48_   _   BIT 1,B|1,C|1,D|1,E|1,H|1,L|1,(HL)|1,A
DD_CB_x1_56  BIT 2,(IX+x)
FD_CB_x1_56  BIT 2,(IY+x)
CB_50_   _   BIT 2,B|2,C|2,D|2,E|2,H|2,L|2,(HL)|2,A
DD_CB_x1_5E  BIT 3,(IX+x)
FD_CB_x1_5E  BIT 3,(IY+x)
CB_58_   _   BIT 3,B|3,C|3,D|3,E|3,H|3,L|3,(HL)|3,A
DD_CB_x1_66  BIT 4,(IX+x)
FD_CB_x1_66  BIT 4,(IY+x)
CB_60_   _   BIT 4,B|4,C|4,D|4,E|4,H|4,L|4,(HL)|4,A
DD_CB_x1_6E  BIT 5,(IX+x)
FD_CB_x1_6E  BIT 5,(IY+x)
CB_68_   _   BIT 5,B|5,C|5,D|5,E|5,H|5,L|5,(HL)|5,A
DD_CB_x1_76  BIT 6,(IX+x)
FD_CB_x1_76  BIT 6,(IY+x)
CB_70_   _   BIT 6,B|6,C|6,D|6,E|6,H|6,L|6,(HL)|6,A
DD_CB_x1_7E  BIT 7,(IX+x)
FD_CB_x1_7E  BIT 7,(IY+x)
CB_78_   _   BIT 7,B|7,C|7,D|7,E|7,H|7,L|7,(HL)|7,A

DC_x2        CALL C,x
FC_x2        CALL M,x
D4_x2        CALL NC,x
C4_x2        CALL NZ,x
F4_x2        CALL P,x
EC_x2        CALL PE,x
E4_x2        CALL PO,x
CC_x2        CALL Z,x|x
3F_          CCF
DD_BE_x1     CP (IX+x)
FD_BE_x1     CP (IY+x)
B8_   _      CP B|C|D|E|H|L|(HL)|A
```

```
FE_x1       CP x
ED_A9       CPD
ED_B9       CPDR
ED_A1       CPI
ED_B1       CPIR
2F          CPL

27          DAA
35          DEC (HL)
DD_35_x1    DEC (IX+x)
FD_35_x1    DEC (IY+x)
3D          DEC A
05          DEC B
0B          DEC BC
0D          DEC C
15          DEC D
1B          DEC DE
1D          DEC E
25          DEC H
2B          DEC HL
DD_2B       DEC IX
FD_2B       DEC IY
2D          DEC L
3B          DEC SP
F3          DI
10_p1       DJNZ p

FB          EI
E3          EX (SP),HL
DD_E3       EX (SP),IX
FD_E3       EX (SP),IY
08          EX AF,AF'
EB          EX DE,HL
D9          EXX

76          HALT

ED_46       IM 0
ED_56       IM 1
ED_5E       IM 2
ED_78       IN A,(C)
DB_x1       IN A,(x)
ED_40       IN B,(C)
ED_48       IN C,(C)
ED_50       IN D,(C)
ED_58       IN E,(C)
ED_60       IN H,(C)
ED_68       IN L,(C)
DD_34_x1    INC (IX+x)
FD_34_x1    INC (IY+x)
3C          INC A
03          INC BC|B
0C          INC C
13          INC DE|D
1C          INC E
23          INC HL|H
DD_23       INC IX
FD_23       INC IY
2C          INC L
33          INC SP|(HL)
ED_AA       IND
ED_BA       INDR
ED_A2       INI
ED_B2       INIR

E9          JP (HL)
DD_E9       JP (IX)
FD_E9       JP (IY)
DA_x2       JP C,x
```

```
FA_x2        JP M,x
D2_x2        JP NC,x
C2_x2        JP NZ,x|x
F2_x2        JP P,x
EA_x2        JP PE,x
E2_x2        JP PO,x
CA_x2        JP Z,x
38_p1        JR C,p
30_p1        JR NC,p
20_p1        JR NZ,p
18_p1        JR p
28_p1        JR Z,p

02           LD (BC),A
12           LD (DE),A
77           LD (HL),A
70           LD (HL),B|(HL),C|(HL),D|(HL),E|(HL),H|(HL),L
36_x1        LD (HL),x
DD_77_x1     LD (IX+x),A
DD_70_x1     LD (IX+x),B|(IX+x),C|(IX+x),D
DD_73_x1     LD (IX+x),E|(IX+x),H|(IX+x),L
DD_36_x1_x1  LD (IX+x),x
FD_77_x1     LD (IY+x),A
FD_70_x1     LD (IY+x),B|(IY+x),C|(IY+x),D
FD_73_x1     LD (IY+x),E|(IY+x),H|(IY+x),L
FD_36_x1_x1  LD (IY+x),x
ED_43_x2     LD (x),BC
ED_53_x2     LD (x),DE
22_x2        LD (x),HL
DD_22_x2     LD (x),IX
FD_22_x2     LD (x),IY
ED_73_x2     LD (x),SP
0A           LD A,(BC)
1A           LD A,(DE)
DD_7E_x1     LD A,(IX+x)
FD_7E_x1     LD A,(IY+x)
78           LD A,B|A,C|A,D|A,E|A,H|A,L|A,(HL)|A,A
ED_57        LD A,I
ED_5F        LD A,R
3A_x2        LD A,(x)
3E_x1        LD A,x
DD_46_x1     LD B,(IX+x)
FD_46_x1     LD B,(IY+x)
40           LD B,B|B,C|B,D|B,E|B,H|B,L|B,(HL)|B,A
06_x1        LD B,x
ED_4B_x2     LD BC,(x)
01_x2        LD BC,x
DD_4E_x1     LD C,(IX+x)
FD_4E_x1     LD C,(IY+x)
48           LD C,B|C,C|C,D|C,E|C,H|C,L|C,(HL)|C,A
0E_x1        LD C,x
DD_56_x1     LD D,(IX+x)
FD_56_x1     LD D,(IY+x)
50           LD D,B|D,C|D,D|D,E|D,H|D,L|D,(HL)|D,A
16_x1        LD D,x
12           LD (DE),A
ED_5B_x2     LD DE,(x)
11_x2        LD DE,x
DD_5E_x1     LD E,(IX+x)
FD_5E_x1     LD E,(IY+x)
58           LD E,B|E,C|E,D|E,E|E,H|E,L|E,(HL)|E,A
1E_x1        LD E,x
DD_66_x1     LD H,(IX+x)
FD_66_x1     LD H,(IY+x)
60           LD H,B|H,C|H,D|H,E|H,H|H,L|H,(HL)|H,A
26_x1        LD H,x
2A_x2        LD HL,(x)
ED_47        LD I,A
DD_2A_x2     LD IX,(x)
```

```
DD_21_x2     LD IX,x
FD_2A_x2     LD IY,(x)
FD_21_x2     LD IY,x
DD_6E_x1     LD L,(IX+x)
FD_6E_x1     LD L,(IY+x)
68           LD L,B|L,C|L,D|L,E|L,H|L,L|L,(HL)|L,A
2E_x1        LD L,x
ED_4F        LD R,A
F9           LD SP,HL
DD_F9        LD SP,IX
FD_F9        LD SP,IY
ED_7B_x2     LD SP,(x)
31_x2        LD SP,x|(x),A
ED_A8        LDD
ED_B8        LDDR
ED_A0        LDI
ED_B0        LDIR

ED_44        NEG
00           NOP

DD_B6_x1     OR (IX+x)
FD_B6_x1     OR (IY+x)
B0           OR B|C|D|E|H|L|(HL)|A
F6_x1        OR x
ED_BB        OTDR
ED_B3        OTIR
ED_79        OUT (C),A
ED_41        OUT (C),B
ED_49        OUT (C),C
ED_51        OUT (C),D
ED_59        OUT (C),E
ED_61        OUT (C),H
ED_69        OUT (C),L
D3_x1        OUT (x),A
ED_AB        OUTD
ED_A3        OUTI

F1           POP AF
C1           POP BC
D1           POP DE
E1           POP HL
DD_E1        POP IX
FD_E1        POP IY

F5           PUSH AF
C5           PUSH BC
D5           PUSH DE
E5           PUSH HL
DD_E5        PUSH IX
FD_E5        PUSH IY

DD_CB_x1_86 RES 0,(IX+x)
FD_CB_x1_86 RES 0,(IY+x)
CB_80       RES 0,B|0,C|0,D|0,E|0,H|0,L|0,(HL)|0,A
DD_CB_x1_8E RES 1,(IX+x)
FD_CB_x1_8E RES 1,(IY+x)
CB_88       RES 1,B|1,C|1,D|1,E|1,H|1,L|1,(HL)|1,A
DD_CB_x1_96 RES 2,(IX+x)
FD_CB_x1_96 RES 2,(IY+x)
CB_90       RES 2,B|2,C|2,D|2,E|2,H|2,L|2,(HL)|2,A
DD_CB_x1_9E RES 3,(IX+x)
FD_CB_x1_9E RES 3,(IY+x)
CB_98       RES 3,B|3,C|3,D|3,E|3,H|3,L|3,(HL)|3,A
DD_CB_x1_A6 RES 4,(IX+x)
FD_CB_x1_A6 RES 4,(IY+x)
CB_A0       RES 4,B|4,C|4,D|4,E|4,H|4,L|4,(HL)|4,A
DD_CB_x1_AE RES 5,(IX+x)
FD_CB_x1_AE RES 5,(IY+x)
```

```
CB_A8        RES 5,B|5,C|5,D|5,E|5,H|5,L|5,(HL)|5,A
DD_CB_x1_B6  RES 6,(IX+x)
FD_CB_x1_B6  RES 6,(IY+x)
CB_B0        RES 6,B|6,C|6,D|6,E|6,H|6,L|6,(HL)|6,A
DD_CB_x1_BE  RES 7,(IX+x)
FD_CB_x1_BE  RES 7,(IY+x)
CB_B8        RES 7,B|7,C|7,D|7,E|7,H|7,L|7,(HL)|7,A
C9           RET
D8           RET C
F8           RET M
D0           RET NC
C0           RET NZ
F0           RET P
E8           RET PE
E0           RET PO
C8           RET Z
ED_4D        RETI
ED_45        RETN
DD_CB_x1_16  RL (IX+x)
FD_CB_x1_16  RL (IY+x)
CB_10        RL B|C|D|E|H|L|(HL)|A
17           RLA
DD_CB_x1_06  RLC (IX+x)
FD_CB_x1_06  RLC (IY+x)
CB_00        RLC B|C|D|E|H|L|(HL)|A
07           RLCA
ED_6F        RLD
DD_CB_x1_1E  RR (IX+x)
FD_CB_x1_1E  RR (IY+x)
CB_18        RR B|C|D|E|H|L|(HL)|A
DD_CB_x1_0E  RRC (IX+x)
FD_CB_x1_0E  RRC (IY+x)
CB_08        RRC B|C|D|E|H|L|(HL)|A
0F           RRCA
ED_67        RRD
1F           RRA
C7           RST 0
C7           RST 00H
CF           RST 08H
C7           RST 0H
D7           RST 10H
D7           RST 16
DF           RST 18H
E7           RST 20H
DF           RST 24
EF           RST 28H
F7           RST 30H
E7           RST 32
FF           RST 38H
EF           RST 40
F7           RST 48
FF           RST 56
CF           RST 8
CF           RST 8H

DD_9E_x1     SBC A,(IX+x)
FD_9E_x1     SBC A,(IY+x)
98           SBC A,B|A,C|A,D|A,E|A,H|A,L|A,(HL)|A,A
DE_x1        SBC A,x
ED_42        SBC HL,BC
ED_52        SBC HL,DE
ED_62        SBC HL,HL
ED_72        SBC HL,SP
37           SCF
DD_CB_x1_C6  SET 0,(IX+x)
FD_CB_x1_C6  SET 0,(IY+x)
CB_C0        SET 0,B|0,C|0,D|0,E|0,H|0,L|0,(HL)|0,A
DD_CB_x1_CE  SET 1,(IX+x)
FD_CB_x1_CE  SET 1,(IY+x)
```

```
CB_C8        SET 1,B|1,C|1,D|1,E|1,H|1,L|1,(HL)|1,A
DD_CB_x1_D6  SET 2,(IX+x)
FD_CB_x1_D6  SET 2,(IY+x)
CB_D0        SET 2,B|2,C|2,D|2,E|2,H|2,L|2,(HL)|2,A
DD_CB_x1_DE  SET 3,(IX+x)
FD_CB_x1_DE  SET 3,(IY+x)
CB_D8        SET 3,B|3,C|3,D|3,E|3,H|3,L|3,(HL)|3,A
DD_CB_x1_E6  SET 4,(IX+x)
FD_CB_x1_E6  SET 4,(IY+x)
CB_E0        SET 4,B|4,C|4,D|4,E|4,H|4,L|4,(HL)|4,A
DD_CB_x1_EE  SET 5,(IX+x)
FD_CB_x1_EE  SET 5,(IY+x)
CB_E8        SET 5,B|5,C|5,D|5,E|5,H|5,L|5,(HL)|5,A
DD_CB_x1_F6  SET 6,(IX+x)
FD_CB_x1_F6  SET 6,(IY+x)
CB_F0        SET 6,B|6,C|6,D|6,E|6,H|6,L|6,(HL)|6,A
DD_CB_x1_FE  SET 7,(IX+x)
FD_CB_x1_FE  SET 7,(IY+x)
CB_F8        SET 7,B|7,C|7,D|7,E|7,H|7,L|7,(HL)|7,A
DD_CB_x1_26  SLA (IX+x)
FD_CB_x1_26  SLA (IY+x)
CB_20        SLA B|C|D|E|H|L|(HL)|A
DD_CB_x1_2E  SRA (IX+x)
FD_CB_x1_2E  SRA (IY+x)
CB_28        SRA B|C|D|E|H|L|(HL)|A
DD_CB_x1_3E  SRL (IX+x)
FD_CB_x1_3E  SRL (IY+x)
CB_38        SRL B|C|D|E|H|L|(HL)|A
DD_96_x1     SUB A,(IX+x)
FD_96_x1     SUB A,(IY+x)
90_          SUB A,B|A,C|A,D|A,E|A,H|A,L|A,(HL)|A,A
D6_x1        SUB x

DD_AE_x1     XOR (IX+x)
FD_AE_x1     XOR (IY+x)
A8_          XOR B|C|D|E|H|L|(HL)|A
EE_x1        XOR x
```

Anhang E: Übersicht aller Editierbefehle von EDT

<table>
<tr><th>Befehl</th><th>Bedeutung</th></tr>
<tr><td>[.+1]</td><td>Leerbefehl (impliziter Druck)</td></tr>
<tr><td>[.]a
<Text>
.</td><td><Text> hinter Zeile anfügen</td></tr>
<tr><td>[.,.]c
<Text>
.</td><td>Zeilen in <Text> ändern</td></tr>
<tr><td>[.,.]d</td><td>Zeilen löschen</td></tr>
<tr><td>e [Datei]</td><td>benannte oder Standarddatei in den Puffer einlesen</td></tr>
<tr><td>f [Datei]</td><td>Standarddateinamen einstellen oder zeigen</td></tr>
<tr><td>[.]i
<Text>
.</td><td><Text> vor Zeile einfügen</td></tr>
<tr><td>[.,.+1]j</td><td>Zeilen zu einer Zeile zusammenfügen</td></tr>
<tr><td>l</td><td>aktuelle Zeilennummer anzeigen</td></tr>
<tr><td>[.,.] m#</td><td>Zeilen verschieben, so daß sie Zeile # folgen</td></tr>
<tr><td>[.,.]p[#]</td><td>Zeilen drucken und Kontext auf # Zeilen setzen</td></tr>
<tr><td>[.]r [Datei]</td><td>Standard- oder benannte Datei hinter Zeile in den Puffer lesen</td></tr>
<tr><td>[.,.]s/Muster/Ers./[g]rep</td><td>das erste oder alle Muster in den Zeilen ersetzen</td></tr>
<tr><td>q</td><td>editieren beenden</td></tr>
<tr><td>v</td><td>automatisches anzeigen ein- oder ausschalten</td></tr>
<tr><td>[1,|]w [Datei]</td><td>Zeilen in angegebene oder Standard-Datei schreiben</td></tr>
<tr><td>[.,|]z</td><td>Editierpuffer solange anzeigen, bis Taste gedrückt wird</td></tr>
</table>

Befehlspräfixe

<table>
<tr><td>[1,|]g/Muster/</td><td>globale Suche nach Zeilen mit Muster</td></tr>
<tr><td>[1,|]x/Muster/</td><td>globale Suche nach Zeilen ohne Muster</td></tr>
</table>

Anhang F: Übersicht aller Formatierbefehle von FMT

In der folgende Übersicht geben Zahlen in eckigen Klammern den Standardwert für den jeweiligen Punktbefehl dar, der genommen wird, wenn keine andere Angabe gemacht wird. Parameter, die nicht in eckigen Klammern stehen, müssen angegeben werden; es existiert kein Standardwert. Ein Fragezeichen steht für ein einzelnes beliebiges Zeichen. `Datei` ist ein Dateiname, `Text` ist beliebiger Text.

Befehl	Bedeutung
`.bc ?`	Pseudo-Leerzeichen einstellen (blank character)
`.bf [1]`	fett (boldface)
`.bp [#+1]`	Neue Seite beginnen (begin page)
`.br`	Neue Zeile beginnen (break)
`.cc ?`	Befehlszeichen einstellen (command character)
`.ce [1]`	zentrieren (center)
`.cu [1]`	durchgehend unterstreichen (continous underline)
`.dw [1]`	doppelt breite Zeichen (double width)
`.ef [Text]`	Fußzeile für gerade Seiten (even-page footer)
`.eh [Text]`	Kopfzeile für gerade Seiten (even-page header)
`.fi`	Zeilen füllen (fill)
`.fo [Text]`	Fußzeile (footer)
`.he [Text]`	Kopfzeile (header)
`.in [0]`	einrücken (indent)
`.it [1]`	kursiv (italicize)
`.ju`	Blocksatz (justify)
`.lm [11]`	linker Rand (left margin)
`.ls [1]`	Zeilenabstand (line spacing)
`.m1 [1]`	Rand 1 (margin 1, oberer Rand zur Kopfzeile)
`.m2 [2]`	Rand 2 (margin 2, zwischen Text und m1)
`.m3 [2]`	Rand 3 (margin 3, zwischen Text und m4)
`.m4 [9]`	Rand 4 (margin 4, unterer Rand zur Fußzeile)
`.mc ?`	Feldbegrenzung in der Datendatei (merge character)
`.mp [2]`	minimaler Platz für Absatz (minimum paragraph)
`.ne [0]`	Zeilen zusammenhalten (need)
`.nf`	nicht füllen (no filling)
`.nj`	kein Blocksatz (no justifying)
`.nu`	nicht unterstreichen (no underlining)
`.of [Text]`	Fußzeile für ungerade Seiten (odd-page footer)
`.oh [Text]`	Kopfzeile für ungerade Seiten (odd-page header)
`.pl [66]`	Seitenlänge (page lenght)

`.po [0]`	Seitenoffset (page offset)
`.pr [Text]`	Eingabeaufforderung (prompt)
`.rm [74]`	rechter Rand (right margin)
`.rs [0]`	Platz reservieren (reserve space)
`.so Datei`	Quelldatei für Text (source)
`.sp [1]`	Leerzeilen (space)
`.sq [0]`	zusammendrücken (squeeze)
`.ti [0]`	zeitweise einrücken (temporary indent)
`.ul [1]`	nicht durchgehend unterstreichen (underline)
`.. [Text]`	Kommentar

Register

JETZT AUF SCHNEIDER-COMPUTERN:

Markt&Technik
Schneider CPC-Software

Turbo Lader

Die Programm-Bibliothek für Turbo Pascal
über 100 Prozeduren und Funktionen in Turbo Pascal Source Code: Bitmanipulation, Sortierverfahren, Splinefunktionen, Fouriertransformation, Regressionsanalyse und vieles mehr.

3" Schneider-Format

DIE PROGRAMM-BIBLIOTHEK FÜR
TURBO PASCAL®

TURBO-Lader-Grundpaket

Das TURBO-Lader-Grundmodul ist eine umfangreiche Programm-Bibliothek für den TURBO-Pascal-Programmierer. Sie umfaßt zahlreiche ausführlich dokumentierte Prozeduren und Funktionen, die der Profi zur schnellen Lösung seiner Programmieraufgaben verwenden kann.

Das TURBO-Lader-Grundpaket erfordert den TURBO-Pascal-Compiler. Es ist lieferbar auf 3"- und 5 1/4"-Disketten und lauffähig auf dem Schneider CPC 464, CPC 664, CPC 6128 und Joyce.

3"-Disk. Best.-Nr. MS 413
5¹/₄"-Disk. Best.-Nr. MS 415

DM 138,-*

TURBO-Lader Business

TURBO-Lader Business umfaßt einen komfortablen Bildschirm-Maskengenerator und eine professionelle Dateiverwaltung. Der Maskengenerator gibt dem Pascal-Programmierer ein Werkzeug zur einfachen Bearbeitung von Bildschirm-Masken in die Hand.

TURBO-Lader Business erfordert den TURBO-Pascal-Compiler und das TURBO-Lader-Grundpaket. Es ist lieferbar auf 3"- und 5 1/4"-Disketten und lauffähig auf dem Schneider CPC 464, CPC 664, CPC 6128 und Joyce.

3"-Disk. Best.-Nr. MS 423
5¹/₄"-Disk. Best.-Nr. MS 425

DM 148,-*

TURBO-Lader Science

TURBO-Lader Science ist eine Sammlung technisch/wissenschaftlicher Funktionen und professioneller statistischer Verfahren für die Bereiche Medizin, Betriebs- und Volkswirtschaft, Technik und Naturwissenschaften.

TURBO-Lader Science erfordert den TURBO-Pascal-Compiler und das TURBO-Lader-Grundpaket. Es ist lieferbar auf 3"- und 5 1/4"-Disketten und lauffähig auf dem Schneider CPC 464, CPC 664, CPC 6128 und Joyce.

3"-Disk. Best.-Nr. MS 433
5¹/₄"-Disk. Best.-Nr. MS 435

DM 189,-*

Übrigens können Sie auch folgende Turbo-Pascal-Produkte für Schneider CPC und Joyce bei Markt&Technik beziehen:
Turbo Pascal 3.0, Turbo Pascal 3.0 mit Grafikunterstützung,
Turbo Tutor (Deutsch), Turbo Tutor (Englisch), Turbo Graphik Toolbox, Turbo Toolbox.

*inklusive MwSt., unverbindliche Preisempfehlung

TURBO Pascal® ist ein Warenzeichen der Borland Inc., USA. TURBO-Lader, TURBO-Lader Business und TURBO-Lader Science sind Warenzeichen der Fa. Lauer & Wallnitz.

Markt&Technik
Schneider CPC-Software

Hans-Pinsel-Straße 2, 8013 Haar bei München

Markt & Technik-Softwareprodukte erhalten Sie in den Computer-Abteilungen der Kaufhäuser und im Computershop.

Small-C
Entwicklungssystem

Das Small-C-Entwicklungssystem ist ein komplettes Entwicklungspaket für das Betriebssystem CP/M mit Editor, Compiler, Assembler, Linker und vielen weiteren Utilities. Alle Programme sind in Small-C geschrieben, und der Quellcode wird mitgeliefert! Dem kundigen Benutzer wird so die Möglichkeit gegeben, sich das Entwicklungssystem nach seinen Wünschen und Erfordernissen zu erweitern und zu modifizieren. Darüber hinaus erhält der Anwender damit eine umfangreiche Sammlung praxisnaher Programme und Tools, die exzellente Beispiele für die effiziente Programmierung in C darstellen. Dadurch ist dieses Produkt eine wertvolle Fundgrube für jeden ernsthaften C-Programmierer. Das Paket besteht aus drei Teilen, die wiederum aus mehreren Komponenten bestehen.

Markt&Technik
Schneider CPC-Software
Hans-Pinsel-Straße 2
8013 Haar

Small-C-Compiler
Small-C ist ein umfangreicher Subset der Sprache C, dessen Qualität durch diese Programme selbst, die alle in dieser Sprache geschrieben wurden, dokumentiert wird. Der Compiler übersetzt C-Programme in 8080-Assembler; zur Übersetzung des Assembler-Codes kann der mitgelieferte Makroassembler, aber auch der Microsoft-Assembler verwendet werden.

Small-MAC:
Assembler und Utilities
Der Makroassembler-Teil besteht aus sechs Programmen. MAC ist der eigentliche Makroassembler. Er arbeitet mit zwei Läufen und erzeugt relokatierbaren 8-Bit-Objektcode im Microsoft-Format. MAC kann mit Hilfe des Programms CMIT an den Befehlssatz der Prozessoren 8080 oder Z80 angepaßt werden. Der Linker LNK verknüpft Objektmodule mit den benötigten Bibliotheksmodulen zu ausführbaren Programmen. Mit dem Loader LGO können Betriebssystemerweiterungen geladen und gestartet werden. Der Bibliotheksmanager LIB verwaltet Bibliotheken mit LNK-kompatiblen Objektmodulen, und DREL erlaubt den Dump von LNK- und LIB-Dateien.

Small-Tools:
Editor und Text-Tools
Small-Tools ist eine komfortable Sammlung von Text-Tools, die einen weiten Bereich der Textverarbeitung abdecken, von der Eingabe von Programmen und Texten (EDIT) über die Erstellung von Serienbriefen und Formatierung von beliebigen Manuskripten (FORMAT) bis zur Rechtschreibüberprüfung (englisch) und Sortierung von ASCII-Dateien (SORT/MERGE). Darüber hinaus stehen noch etliche Hilfsprogramme zur Verfügung, die kleinere Aufgaben erledigen, jedoch kombiniert eingesetzt werden können und so zu einem extrem leistungsfähigen Tool werden.

Hardware-Voraussetzungen
Das Small-C-Entwicklungssystem benötigt einen Schneider-Computer mit mindestens 56 KByte Speicher und einem Diskettenlaufwerk. Bei den Modellen CPC 464 und CPC 664 ist eine Speichererweiterung notwendig.